UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Demostrador arquitectura publish/subscribe con MQTT

Tesis final de grado

Publicado por la facultad

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

por

Francesc Moreno Cerdà

Cumplimento parcial de los requisitos de

(Grado en ingeniería Telemática) **INGENIERÍA**

Advisor: Anna Calveras Augé

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

telecom BCN

Barcelona, Enero 2018

Abstract

Nowadays, the information and communication are becoming more important in the society. Because of that, the concept of internet of things it's becoming more important.

In this project I'm going to create a demonstrate IoT system that communicates using the protocol MQTT. The document will how the characteristics and functionalities of the protocol. The project it's going to be implement in a different technology to show the adaptability of the protocol to different hardware, software and programing languages.

The document will show that MQTT it's an easy protocol to implement and have a lot od possibilities.

Resum

En les ultimes desenes es pot apreciar com la informació y la comunicació tenen un paper més rellevant dintre de la societat. Per aquest motiu el concepte de internet de las coses (IoT) esta agafant pes dia a dia.

En aquest projecte es realitza un demostrador de un sistema IoT que treballa amb el protocol MQTT, en el que es mostraran les seves funcionalitats y característiques. Per la realització d'aquest projecte s'utilitzaran dispositius de diferents tecnologies per mostrar la gran versatilitat que te el protocol en quant a hardware, software i llenguatges de programació.

Mostrarà com el protocol MQTT es un protocol senill d'implementar y que presenta molta adaptabilitat.

Resumen

En las últimas décadas se ha podido apreciar como la información y la comunicación tienen un papel muy importante dentro de la sociedad. Por este motivo el concepto del Internet de las cosas (IoT) está ganando peso día a día.

En este proyecto se va a realizar un demostrador de un sistema IoT utilizando el protocolo MQTT, en el que se mostrarán las funcionalidades y características de dicho protocolo. Para la realización del proyecto se utilizarán dispositivos de diferentes tecnologías para mostrar la gran adaptabilidad que tiene este protocolo a diferentes hardware, softwares y lenguajes de programación.

Se mostrará como MQTT es un protocolo sencillo de implementar y que presenta una gran adaptabilidad.

Agradecimientos

Tengo que agradecer a mi supervisora del proyecto Anna Calveras Augé la ayuda y dedicación que me ha prestado a la hora de poder realizar este proyecto. Es el primer gran proyecto al que me enfrento solo y sin su ayuda, trabajo y paciencia no habría sido posible. Cuando surgían dudas, siempre sabia mostrarme el camino sin llegar nunca a ser muy intrusiva.

Tengo que estarle muy agradecido ya que también me ha ayudado en tareas que no eran de su función como es la revisión de documentos escritos.

Historial de revision y aprovación

Revisión	Fecha	Propuesta
0	dd/mm/yyyy	Document creation
1	dd/mm/yyyy	Document revision

DOCUMENT DISTRIBUTION LIST

Nombre	e-mail
Francesc Moreno Cerdà	Morenocerda91@gmail.com
Anna Calveras Augé	anna.calveras@entel.upc.edu

Escrito por:		Revisado y aprobado por:	
fecha	25/01/2018	Fecha	dd/mm/yyyy
Nombre	Francesc Moreno Cerdà	Nombre	Zzzzzzz Wwwwwww
Posición	Autor del proyecto	Posición	Project Supervisor

Índice

Listado de Imagines:

Listado de Tablas:

1. Introducción

En las últimas décadas, se ha podido apreciar como la información y la comunicación están teniendo cada vez un papel más importante en todos los ámbitos, tanto a nivel social, económico como ambiental. Este fenómeno viene dado en gran medida por los avances de la tecnología, el desarrollo de internet y de las posibilidades que ofrece.

El internet de las cosas (IoT) es un concepto que se basa en la interconectividad de objetos dotándolos de conexión a internet, siguiendo esta tendencia de obtener información y comunicarse. En el día a día, existen incontables dispositivos que están constantemente captando información y la idea del IoT es dotar esos dispositivos de conexión a internet para poder comunicarse y obtener un beneficio de ello, y este, no tiene por qué ser explícitamente económico.

Siguiendo la tendencia de los últimos años en cuanto a la información y comunicación, junto al concepto de IoT de dotar de internet a los objetos cotidianos, hace evidente que el internet de las cosas está destacando como una de las ramas más importantes y con más proyección dentro de las tecnologías de la información y comunicación (TIC).

El internet de las cosas se está aplicando a ámbitos de medio ambiente; para controlar niveles de contaminación en aire, controlar contaminación del agua en ríos causado por el vertido de fluidos de las fábricas y sensores para detectar fuegos en zonas de bosque con alta probabilidad de incendios. Otro sector donde se aplica es en las ciudades con el objetivo de dar facilidades a los ciudadanos y mejorar los servicios disponibles. Las ciudades que aplican el IoT son conocidas como Smart Cities e incluyen: sistemas de detección de aparcamientos libre, sensores estructurales para controlar el estado de edificios, puentes, monumentos, sistemas para controlar el tráfico reduciendo los atascos y advirtiendo de accidentes, etc. A nivel económico también se está implantando el IoT en empresas, ayudando a mejorar el rendimiento y la producción. La industria 4.0 es un buen ejemplo de ello, y consiste en aplicar el internet de las cosas en los procesos de producción para tener un mejor control de dicha producción en todo momento.

De la misma manera que hay muchos sectores o ámbitos donde se aplica el IoT, también hay muchos protocolos que se utilizan para comunicar los distintos dispositivos de un sistema IoT. Ejemplos de ellos son Advanced Message Queuing Protocol (AMQP), Constrained Application Protocol, Message Queue Telemetry Transport (MQTT), etc.

Este documento no se va a centrar en un sector donde se aplican los conceptos IoT como podrían ser los anteriormente citados, sino que la intención es dentro del gran abanico de protocolos que existen para la comunicación en el internet de las cosas, centrarse en uno en concreto. Por tanto, en este proyecto se va a llevar a cabo un demostrador de un sistema IoT utilizando el protocolo Message Queue Telemetry Transport (MQTT).

MQTT es un protocolo ligero para comunicación de mensajes entre dispositivos que no requieran mucho procesado y no requieran un gran ancho de banda para ello. Se sitúa por encima del protocolo TPC en el modelo OSI, un protocolo extendido en muchos ámbitos y que se basa en el método de comunicación publish/subscribe. Esta última característica es fundamental en este protocolo y es uno de los valores diferenciales conforme a otros protocolos diseñados para IoT. Esto dota a los sistemas que usan MQTT de una estructura diferente en la que las “cosas” no se comunican directamente en modo cliente servidor (como la mayoría de casos del protocolo TCP/IP), sino que se comunica máquina a máquina mediante la publicación y suscripción a tópicos a través de un elemento central

denominado Bróker. El tópico es un UTF8 String, que sirve para que el bróker pueda filtrar los mensajes que publican los clientes para hacerlos llegar a los que están suscritos al tópico. Por tanto, para comunicarse los diferentes clientes, solo es necesario que conozcan la información de ese bróker y el tópico, sin tener que conocer la de los otros elementos del sistema. Más adelante se explicará detalladamente la arquitectura del sistema.

1.1. Descripción del sistema

En este proyecto se lleva a cabo el diseño e implementación de un sistema IoT utilizando el protocolo MQTT. Como se ha comentado en la introducción los dispositivos que obtienen información y la transmiten son muy diversos, por lo que los sistemas IoT tiene que presentar una gran adaptabilidad para poder comunicar elementos distintos, con tecnologías diferentes, canales de comunicación diferentes, plataformas y lenguajes de programación diferentes.

Por esta razón en este proyecto se diseña e implementa un sistema en la que interactúan dispositivos con diferentes tecnologías, canales de comunicación y lenguajes de programación, mostrando que el protocolo MQTT puede cumplir esta característica necesaria para los sistemas IoT.

MQTT es un protocolo publish/subscribe basado en tópicos en el que los clientes envían y reciben mensajes que son gestionados por un elemento central llamado bróker. Por tanto, el protocolo MQTT presenta una tipología de red en estrella.

El sistema realizado consta de 3 clientes y un bróker. Los 3 clientes responden a diferentes funcionalidades que generalmente presentan los sistemas IoT: elementos que captan información y la transmiten, elementos de gestión y elementos que muestran la información requerida. Esto no quiere decir que un mismo dispositivo no pueda realizar varias de ellas a la vez, y esta es una de las características del protocolo MQTT, que mediante la publicación y la suscripción a tópicos un mismo dispositivo puede realizar varias a la vez.

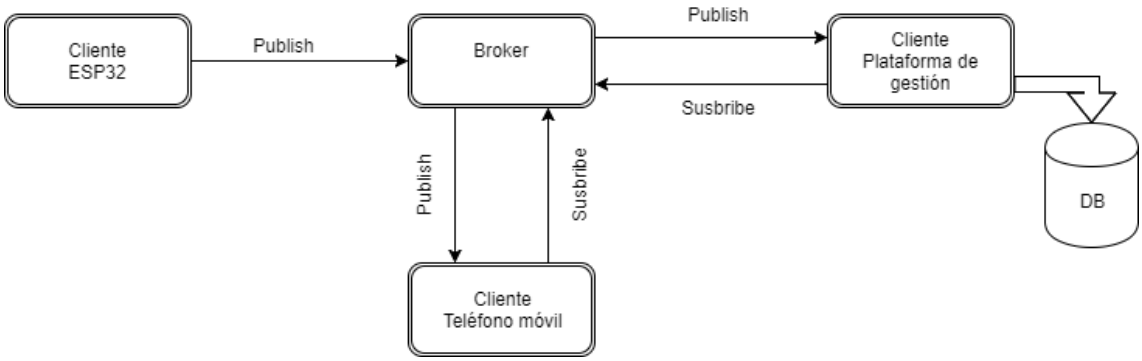
Un primer cliente se implementa sobre un dispositivo ESP32, que es un chip integrado con un microprocesador, dotado de conexión wifi y bluetooth y que permite la adhesión de sensores externos. Este dispositivo se encargará de medir la temperatura y la pulsación de un botón y transmitirla al sistema. Recibirá los mensajes del sistema una vez ha sido procesada la temperatura que envía.

El segundo cliente se implementa sobre un ordenador portátil con SO Linux, con Ubuntu 16.04 LTS, que hará la función de gestión del sistema, guardado de los datos obtenidos en base de datos y envío de los mensajes de control o advertencia. Los mensajes de advertencia que manda al sistema son producidos al comprobar que la temperatura supera un límite establecido alertando al dispositivo móvil, otro elemento de nuestro sistema. En nuestro sistema la plataforma de gestión actúa como publicador y subscriptor.

Por último, el sistema contará con un dispositivo móvil o Smartphone que obtendrá datos del dispositivo ESP32 y los mostrará al usuario final. También recibirá los mensajes de la plataforma de gestión para mostrarlos al usuario. En la plataforma móvil no se encarga

únicamente de recibir datos, sino que también transmitirá la localización del dispositivo al sistema.

El sistema resultante se muestra en la figura (1).



1.2. Requerimientos y especificaciones del sistema

Antes llevar a cabo la implementación del proyecto se marcan unos requerimientos que deberá cumplir el sistema.

- Creación de una arquitectura publish/susbribe usando el protocolo MQTT.
- El sistema tiene que contar con:
  - Clientes: se han de implementar en 3 tecnologías diferentes.
    - Un dispositivo SoC (System on Chip). Para el proyecto hemos seleccionado el ESP32 por sus características en las que permite la integración de sensores y dispone de antena WiFi para transmitir.
    - Un terminal móvil (Android).
    - Plataforma de gestión (Ordenador con Ubuntu)
  - Bróker: Este bróker se ha de crear en 2 plataformas distintas (no se usarán a la vez, sino que una sustituye a la otra).
    - Ordenador de sobremesa con Ubunut 16.04.
    - Respberry con Raspbian.
- El dispositivo ESP32 ha de ser capaz de:
  - Medir 2 o más variables de estado (ej. Temperatura, botón)
    - Mediante sensores internos.
    - Mediante sensores externos.
  - Transmitir datos mediante WI-FI.
  - Conectarse al sistema(bróker).
  - Capacidad de publicar mensajes con el protocolo Publish/Subscribe MQTT.
- Terminal móvil ha de ser capaz de:
  - Conectarse al sistema (bróker) con el protocolo Publish/Subscribe MQTT.
  - Medir localización.
  - Medir la hora.
  - Publicar mensajes al sistema con el protocolo Publish/Subscribe MQTT.
  - Suscribirse a tópicos en el sistema para recibir mensajes con el protocolo Publish/Subscribe MQTT.
- Plataforma de gestión ha de ser capaz de:

- Subscribirse a tópicos en el sistema para recibir mensajes con el protocolo Publish/Subscribe MQTT.
- Publicar mensajes al sistema con el protocolo Publish/Subscribe MQTT.
- Guardar en bases de datos (sencilla) la información que se considere importante.

- Bróker: Se ha de poder crear en 2 entornos diferentes. (que no actúan a la vez)
  - En un ordenador.
  - En una Raspberry.

Especificaciones:

- QoS: Utilización de diferentes calidades de servicio asignadas a importancia de mensajes en el sistema.
- ESP32:
  - Programado en lenguaje arduino.
  - Temperatura: cada 5 segundos máximo, usando un sensor externo. Usaremos el DHT11 por ser un sensor digital que permite calcular la humedad y temperatura a la vez.
  - Controlará si ha sido pulsado el botón, se usará un botón integrado en la placa.
  - Se estudiará la posibilidad de añadir más sensores.
  - Publicará estos datos en tópicos separados.
- Móvil, aplicación Android.
  - Aplicación diseñada para Android 5.0 o superior.
  - Conectará con google play services para obtener datos de localización.
  - Obtendrá la hora y fecha con funciones nativas de Android.
  - Generación de notificaciones al recibir mensajes del sistema.
  - Presentación de 2 pantallas de interfaz de Usuario.
  - Publicación localización al sistema mediante un tópico.
  - Suscrición a tópico de ESP32 y del cliente Python.
- Plataforma de gestión:
  - Será programada en Python.
  - Será capaz de suscribirse a tópicos del ESP32 y del móvil.
  - Guardará los datos de posición y temperatura en una base de datos. Se ha seleccionado MySQL por su fácil interacción en distas plataformas y una de ellas es Python mediante el módulo MySQLdb.
  - Estos datos estarán guardados en tablas.
  - Podrá publicar mensajes al sistema mediante un tópico.
- Bróker:
  - Tanto para Raspberry como para Ubuntu se utilizará el bróker mosquitto 1.4.14
  - Raspberry:
    - Se usará una Raspberry pi2
    - Sistema operativo Raspbian
  - Ordenador sobremesa:
    - Se usará un equipo con Ubuntu 16.04
    -

1.3. Objetivos

Este proyecto pretende ser un documento explicativo donde mostrar las características, funcionalidades, puntos fuertes del protocolo MQTT y exponer una implementación de un sistema, de forma detallada, para que otras personas o entidades puedan informarse, reproducirlo o trabajar a partir de él.

También pretende mostrar la gran aplicabilidad del protocolo a diferentes tecnologías, softwares, herramientas de desarrollo y lenguajes de programación.

1.4. Plan de trabajo

El sistema propuesto en este documento consta de varios elementos o plataformas que interactúan entre ellas y como comentábamos hay un elemento central que es el bróker que gestiona todos los mensajes. Por esta razón el elemento en el que se ha de empezar a trabajar para la ejecución del proyecto es el bróker.

Una vez el bróker esté instalado, configurado y comprobado que funcione correctamente, se puede empezar a trabajar en los clientes. En caso de contar con un equipo, formado por varias personas, se puede proceder al desarrollo de los 3 clientes simultáneamente. Si es verdad que las 3 plataformas deben comunicarse entre ellas, si en algún momento de la implementación se necesita testear el sistema se puede hacer ya que en el bróker se instala también un cliente para poder testear que el sistema funciona correctamente.

Por el contrario, no se dispone de un equipo, se deberán desarrollar las 3 plataformas de cliente una detrás de la otra, sin importar el orden por el cual empezar o terminar.

Finalmente, una vez funcionan todos los elementos del sistema por separado ya se puede proceder a hacer los test de funcionamiento con todos los elementos y posterior análisis del sistema.

1.5. Incidencias

Durante la realización del proyecto han aparecido diferentes incidencias o dificultades que se han ido superando con mayor o menor complicación. Pero hace falta hacer hincapié en un punto donde aparecieron bastantes problemas que retrasaron los planes de trabajo diseñados al principio y requirieron más tiempo del previsto.

El módulo del esp32 ha generado problemas de tipo eléctrico y manual. Se utiliza un sensor digital (DHT11) que puede medir temperatura y humedad conectándose al chip. El sensor DHT11 tiene un rango dinámico de alimentación entre 3V y 5,5V. Nuestro ESP32 tiene varios pines que generan una salida de 3,3V, por lo que el sensor no tendría que tener ningún problema para trabajar correctamente. Cuando conectamos el sensor y ejecutamos el código empiezan los problemas. Nuestro módulo ESP32 junto con el sensor empieza a fallar de forma aleatoria. En algunas ocasiones al encenderse no conecta a la WiFi, siendo este el primer paso. En otras, el sensor no obtiene la información necesaria, por lo que no envía la información al bróker. Y en los momentos en los que funcionaba bien, si calentábamos el sensor con las manos o aliento volvía a fallar.

Por tanto, realizamos pruebas para llegar a la raíz del problema. Se revisaron las soldaduras del ESP32 y se reemplazaron los cables de conexión al sensor, pero siguió con el mismo comportamiento. Entonces se buscó otro dispositivo ESP32 y se instaló todo lo necesario, pero tampoco mejoró.

En este momento se plantearon alternativas como la de usar sensores internos del dispositivo que tomaran medidas, como la temperatura interna del chip o un sensor magnético. Esta solución se llegó a implementar para poder continuar la construcción de todo el sistema, dado que para la realización del demostrador era más prioritario la comunicación y funcionamiento de todo el sistema, a las medidas que obtenía.

Persistían los problemas de inestabilidad al calentar el sensor. Al final se llegó a la conclusión que la manera de calentar no era la correcta ya que el sensor también tiene un sensor de humedad y alcanzaba valores críticos superiores al 90% de humedad que especifican en el datasheet del sensor. Por tanto, se eliminó el programa que medía humedad y para las pruebas se usaba otro método para calentar el sensor. Finalmente, el funcionamiento fue correcto y se pudo integrar el sensor en nuestro sistema.

Este imprevisto, que puede ser trivial implicó un retraso en poder probar todo el funcionamiento del sistema una vez los otros módulos funcionaban por separado y por eso se recurrió a no usar el sensor para poder seguir avanzando. Pero posteriormente, cuando ya funcionaba el sistema se retomaron las pruebas y se puedo solventar.

2. Antecedentes

Como se comenta en la introducción en los sistemas IoT conviven una gran cantidad de dispositivos diferentes, con hardware diferentes, softwares diferentes, canales de comunicación diferentes, pero la gran mayoría de ellos coinciden en que son dispositivos con capacidades limitadas. Las limitaciones vienen tanto a nivel económico, capacidad de procesado, canales de comunicación, y de batería.

Los protocolos IoT han de adecuarse a estas necesidades que presentan los sistemas y han de tener las siguientes características:

A) Permitir una gran escalabilidad debido al gran número de dispositivos.
B) Disponer de mecanismos para asegurar la comunicación en los entornos donde el canal no sea fiable.
C) Minimizar la sobrecarga de las cabeceras de los paquetes para consumir el mínimo ancho de banda.
D) Mínimo consumo a nivel de procesado y batería.

2.1. Modelo Publish/Subscribe

Este modelo de comunicación presenta una alternativa al típico modelo cliente/servidor donde el cliente se comunica directamente con el punto final. Por el contra, el modelo publish/subscribe se basa en la comunicación máquina a máquina, en el que un cliente envía un mensaje particular (publicador) a otro cliente, o varios, que reciben el mensaje (suscriptor). Esta comunicación se lleva a cabo sin que los clientes tengan conocimiento el uno del otro, ya que hay un tercer elemento en esta comunicación que es el bróker. El bróker es el elemento conocido tanto por el publicador como por el subscriptor y es el encargado de redirigir los mensajes acordemente.

El hecho de que los clientes que publican y reciben los mensajes no tengan conocimiento uno de otros dota a los sistemas publish/subscribe de:

- Desacoplamiento en el tiempo, por lo que no tienen que ejecutarse a la vez.
- Desacoplamiento de sincronismo, por lo que los clientes no tienen que detener las operaciones que estén realizado.

Por estos motivos el modelo de comunicación publish/subscribe se adapta en gran medida a estos requerimientos que suelen presentar los sistemas de internet de las cosas, ejemplo de ello es que muchos de los protocolos IoT se basan en él.

2.2. MQTT

Message Queue Telemetry Transport (MQTT) es un protocolo de comunicación de mensajes basado en el método publish/subscribe, situado por encima del protocolo TCP. Es un protocolo ligero y fácil de implementar que requiere pocos recursos a nivel de procesamiento y ancho de banda.

Al seguir el modelo publish/subscribe los clientes no han de ser conocedores unos de otros por lo que cada cliente solamente ha de conocer la dirección y el puerto del bróker. Esto hace que la comunicación entre clientes sea asíncrona dando como resultado una comunicación denominada near-realtime. El bróker será el encargado de redirigir los mensajes a los diferentes clientes según la subscripción a tópicos.

Los tópicos son string UFT-8, que usa el bróker para dirigir los paquetes entre los diferentes clientes conectados a él. Cuando un cliente publica un mensaje a un tópico el bróker es el encargado de enviar ese mensaje a todos aquellos clientes que estén suscritos.

El tópico debe contener como mínimo un carácter y puede tener o no distinción de niveles, que se separan mediante "/", dando una estructura jerárquica a los tópicos. Hace falta recalcar que el tópico hace distinciones entre mayúsculas y minúsculas. Unos ejemplos podrían ser:

1)"miCasa/primerPiso/luz"

2)"miCasa/segundoPiso/luz"

3)"miCasa/primerPiso/temperatura"

Existen 2 símbolos comodín que son el + y el #. El + actúa como comodín a un nivel por lo que si alguien se subscribe a "miCasa/+/luz" recibiría mensajes del primer y segundo tópico. En cambio, el # actúa como comodín de forma multinivel, por lo que si se subscriben a "miCasa/#" recibiría los mensajes de todos los tópicos.

El protocolo MQTT también implementa calidades de servicio (QoS) para poder tener confirmación de la entrega de paquetes. Existen QoS:

- QoS 0, denominada fire and forget, en la que no se realiza ningún tipo de comprobación, el mensaje es mandado solo una vez.
- QoS 1, denominada at laest one, en la que se requiere un reconocimiento, pero que como dice su nombre puede llegar a enviarse más de una vez.
- Qos 2, denominada exacly one, en la que hay un mecanismo de comprobación por la cual solo se entrega exactamente una vez el mensaje sin repeticiones

El protocolo MQTT presenta 13 diferentes paquetes de control con el que lleva a cabo toda la comunicación: CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, PINGREQ, PINGRESP y DISCONNECT. Estos serán analizados en el apartado de resultados. Todos los mensajes del protocolo MQTT tienen como mínimo 2 Bytes de cavecera fija en la cual se indica: el tipo de mensaje, si esta duplicado, la calidad de servicio del mensaje y si se tene que guardar. Después de la cabecera fija, le sigue una variable que dependerá de cada uno de los mensjaes y finalmente el contenido del mensaje.

MQTT implementa otro típo de funcionalidades como pueden ser el retain message, o mensaje retenido. Este mensaje es guardado, junto con el tópico, por el bróker y cuando un cliente se subscribe a ese tópico el bróker le manda en primera instancia ese mensaje. (configuración)

Otra funcionalidad a destacar en el protocolo MQTT, y muy interesante para los sistemas IoT, es el last will o último deseo. Este es un mensaje que se enviará en caso futuro como indica su nombre. Si un dispositivo ha activado este will message y se desconecta

repentinamente sin mandar el mensaje de desconexión al bróker, este último mandará ese mensaje a los suscritores.

Para detectar esa desconexión, el bróker tiene un parámetro keep alive que si en un número determinado de segundos no recibe comunicación con el dispositivo detecta que se ha desconectado.

Como ultima funcionalidad a destacar es la de persistent session o sesión persistente, en la que si esta activa, tanto el cliente como el servidor han de mantener un estado de sesión. Este estado implica:

- Cliente:
  - Tiene que conservar los mensajes con QoS 1 y QoS2 que no han sido confirmados por el bróker.
  - Y los mensajes con QoS2 que aún no ha confirmado al bróker.
- Broker:
  - Las suscriciones del cliente
  - Los mensajes con QoS 1 y QoS 2 que no han sido completamente confirmados por el cliente.
  - Los mensajes con QoS 1 y QoS 2 pendientes de ser transmitidos al cliente.

En cuanto al apartado de seguridad el protocolo MQTT puede requerir un usuario y una contraseña para que los clientes se puedan conectar al bróker. Esto junto con el TLS/SSL permiten crear un sistema muy seguro. Pueden actuar por separado, o incluso no tener mecanismo de autenticación ni de encriptado.

Por todas estas características que tiene el protocolo MQTT hace que sea uno de los más idóneos a la hora de realizar la comunicación de mensajes en los sistemas IoT. Muestra de ello es que muchos de los sistemas cloud para IoT de grandes empresas como google (google cloud IoT) y Amazon (AWS) son compatibles o utilizan el protocolo MQTT para realizar la comunicación de datos.

Del párrafo anterior, junto con la gran cantidad de SDK, API y librerías que existen del protocolo MQTT ha sido uno de los motivos por los cuales se ha escogido este protocolo para llevar a cabo este proyecto.

2.3. Otros Protocolo IoT

COAP

Constrained Application Protocol es un protocolo de comunicación síncrono que se encuentra en la capa de aplicación dentro del modelo OSI. Es un protocolo diseñado por Internet Engineering Task Force (IETF) basado en la metodología request/response con el objetivo de ser un protocolo ligero para poder utilizarse dentro del mundo de internet de las cosas donde los dispositivos no son muy potentes y no es conveniente que usen mucho ancho de banda. Por esta razón el COAP está implementado sobre UDP para evitar la sobrecarga y el tráfico que genera el protocolo TCP.

Al estar implementado sobre UDP, que no tiene mecanismos de reconocimiento de mensajes y retransmisiones, COAP tiene diseñado un control de menajes a través de

calidad de servicio (QoS). La calidad de servicio viene expresada dentro de la cabecera de cada paquete ocupando 2 bits y en ella se especifica si los mensajes transmitidos requieren confirmación por parte del receptor o no.

El protocolo HTTP es el protocolo más utilizado y común a nivel de comunicación basado en request/response, y es por esta razón que el COAP está diseñado en base a este. Por este motivo usa algunos métodos que están definidos en el HTTP como son el GET, PUT, POST y DELETE. Por esto, el protocolo COAP tiene una gran compatibilidad con el protocolo HTTP, siendo este un punto importante a tener en cuenta a la hora de escoger un protocolo para un sistema IoT.

Una característica importante del COAP es que soporta multicast, siendo beneficioso para los sistemas IoT donde suele haber un gran número de dispositivos conectados.

Para finalizar con el protocolo COAP trataremos el tema de la seguridad. Este protocolo no incluye ningún mecanismo de seguridad por sí mismo, pero si fuera necesaria para nuestra aplicación se podría utilizar el protocolo Datagram Transfer Layer Protocol (DTLS) que corre por encima de UDP. Esto lastra un poco el protocolo ya que incrementaría el consumo de ancho de banda del sistema.

XMPP

El protocolo Extensible Messaging and Presence Protocol (XMPP) fue diseñado hace más de una década para la comunicación chat e intercambio de mensajería. Una característica importante de este protocolo, y que lo diferencia de los demás protocolos de IoT, es la capacidad de trabajar con el modelo publish/subscribe y con el modelo request/response.

XMPP es un protocolo que corre por encima de TCP en la escala OSI, y que es una extensión basada en el XML. Esto puede resultar poco conveniente para algunos sistemas IoT ya que el XML, como el HTML, usa en su código etiquetas probocando una sobrecarga de los mensajes innecesaria, y también aumentado el procesado.

Por último, hace falta recalcar que XMPP no tiene calidad de servicio inherente al protocolo, sino que la retransmisión de datos y fiabilidades viene dada por la del protocolo TCP sobre el que corre.

AMQP

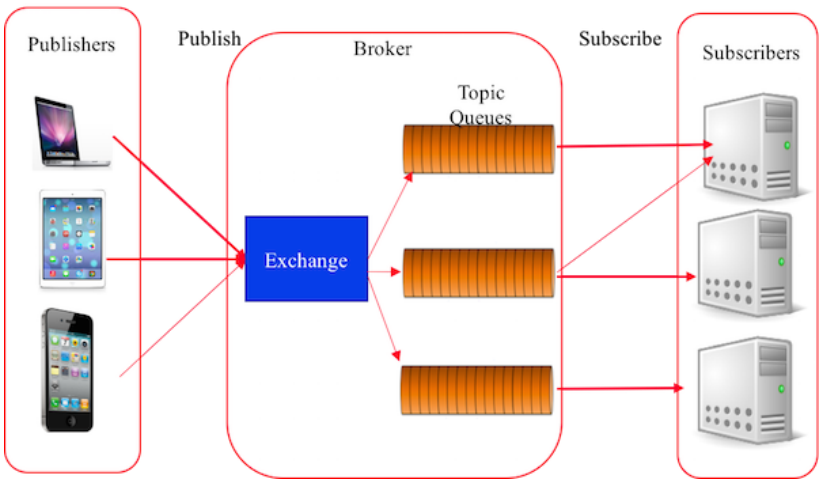
El protocolo Advanced Message Queuing Protocol (AMQP) proviene del mundo de las finanzas. Dicho protocolo puede usarse sobre diferentes protocolos de transporte, pero sobretodo se usa sobre el TCP.

Es un protocolo que en muchos aspectos y funcionamientos es parecido al MQTT. Como el MQTT, este protocolo tiene 3 QoS que son las mismas, y que se podrían definir como; el mensaje se envía una sola vez, llega una o más veces (pudiendo ser más) o llega exactamente una vez. Otra característica que también comparten el MQTT y el AMQP es en cuanto a la seguridad que los 2 utilizan el protocolo TLS/SSL.

Pero si hay una ventaja de este protocolo que lo diferencia del MQTT es la posibilidad guardar los mensajes para después enviarlos garantizando la fiabilidad de la comunicación incluso después de interrupciones de la red.

Este protocolo presenta mejores rendimientos cuando los anchos de banda del sistema son mayores y cuando hay que transmitir a muchos usuarios.

Una empresa que utiliza este protocolo es JPMorgan, para mandar alrededor de 1 billón de mensajes por día. Figura (2)



WEBSOCKET

El protocolo Websocket fue diseñado como iniciativa por parte de HTML 5 para facilitar las comunicaciones sobre TCP. Este protocolo no se puede considerar un protocolo publish/subscribe o request/response como los anteriores, pero sí que tiene ciertas características o funcionalidades heredadas.

El protocolo websocket establece una conexión con el servidor parecida al protocolo HTTP, pero una vez establecida esta conexión, crea un canal full-duplex para el intercambio de mensajes asíncronos entre los dispositivos. Esta conexión permite reducir la sobrecarga de las comunicaciones eliminado parte de las cabeceras que hay en las conexiones normales en internet.

Como los protocolos comentados anteriormente, al correr sobre TCP puede utilizarse el protocolo TLS/SSL para la transmisión segura y cifrada en caso de que el sistema lo requiriese.

Esta forma de comunicación no es la más indicada si se pretende crear una red con muchos dispositivos, ya que implica tener que crear muchas conexiones full-dúplex. También este establecimiento y mantenimiento del canal provoca que los dispositivos que lo usan tengan una mayor carga de procesado y consumo de batería. Es por esta razón que no es conveniente usarse en sistemas IoT con dispositivos de bajo consumo.

Para terminar con los antecedentes de este documento, mostramos una tabla comparativa con algunos de las características más importantes de los protocolos a modo de comparación. Tabla (1).

Protocolo	QoS	Transporte	Seguridad	Arquitectura
MQTT	Si	TCP	TLS/SSL	Publish/Subscribe
CoAP	Si	UDP	DTLS	Request/Response
XMPP	No	TCP	TLS/SSL	Publish/Subscribe Request/Response
AMQP	Si	TCP	TLS/SSL	Publish/Subscribe
Web Socket	No	TCP	TLS/SSL	Client/Server

3. Desarrollo del proyeto

En este apartado del documento se lleva a cabo una descripción más precisa y técnica del sistema que la inicialmente presentada en la introducción y donde se muestra exhaustivamente todos los procesos realizados para la implementación del sistema paso a paso. Esto incluye datos de los dispositivos utilizados, funcionalidad del sistema, instalaciones de software y explicación del código utilizado.

En general, en los sistemas IoT los diferentes elementos, dispositivos o plataformas suelen agruparse en 3 funcionalidades; obtención de información y transmisión, gestión o análisis de la información y mostrado de la información o actuación. Esto no quiere decir que un mismo dispositivo no pueda realizar varias de ellas a la vez, y esta es una de las características del protocolo MQTT, que mediante la publicación y la suscripción a tópicos un mismo dispositivo puede realizar varias a la vez.

El sistema diseñado en este proyecto consta de 4 plataformas que interactúan a la vez: el bróker, y 3 clientes, en los que se reproduce un sistema típico de IoT. También, cada cliente del sistema está implementado en dispositivos con tecnologías diferentes, canales de comunicación diferentes y lenguajes de programación, para así demostrar que el protocolo MQTT puede presentar una gran adaptabilidad a todo ello.

3.1. Funcionamiento del Sistema

El ESP32 mide la temperatura ambiente mediante un sensor externo (DHT11), cada 2 segundos y se publica con el tópico temperatures con QoS 0. A su vez, también comprueba si se ha pulsado el botón integrado en la placa y en caso afirmativo publica un mensaje al tópico alerts con la palabra button como datos.

El bróker recibe estos mensajes y los manda a los subscriptores de los tópicos.

La plataforma de gestión está suscrita al tópico temperatures, por lo que recibirá el mensaje del ESP32 con la temperatura obtenida y la comparará con una temperatura máxima de 30 grados. Tanto si esta temperatura es superior como inferior la aplicación procede a guardarla en la base de datos. De esta forma el sistema tiene un registro de todas las temperaturas obtenidas por el ESP32. En caso de que la temperatura sea superior, la plataforma de gestión ha de informar tanto al ESP32 como al dispositivo móvil publicando 2 mensajes, iguales, pero uno en el tópico alerts y otro en el tópico action.

El bróker recibe estos mensajes y los manda a los subscriptores.

El ESP32 está suscrito al tópico action y cuando recibe un mensaje, enciende el led que tiene integrado la placa.

Por otro lado, tenemos al dispositivo móvil que mediante google play services accede a la localización del teléfono cada 10 segundos (no es exacto) y publica un mensaje con la latitud, longitud y la hora al tópico location. También está suscrito tanto al tópico temperatures como al tópico alerts. Cuando recibe un mensaje al tópico temperatures (proviene del ESP32) el teléfono muestra la temperatura en un toast. Si recibe un mensaje en el tópico alerts el teléfono genera una notificación que trasciende al sistema operativo.

Si se clica sobre la notificación carga otra pantalla en la que expresa a que es debida esta alerta, si ha sido porque en el ESP32 se ha pulsado el botón o es la plataforma de gestión la que informa de que la temperatura es muy elevada.

Los mensajes publicados al tópico location son recogidos por el bróker y este lo envía a la plataforma de gestión que es el único cliente que está suscrito a este tópico. El programa de la plataforma de gestión guarda los datos de latitud, longitud y hora en una base de datos.

El protocolo MQTT tiene la funcionalidad de last will, en la que el bróker envía un mensaje a un tópico determinado en caso de que detecte que se ha desconectado el dispositivo. Nuestro ESP32 implementa esta funcionalidad en su código por lo que si desconecta del sistema el bróker al cabo de 15 segundos sin recibir mensajes del ESP32 detectará que se ha desconectado y por tanto mandará un mensaje al tópico alerts diciendo que se ha desconectado el ESP32. El dispositivo móvil recibirá este mensaje y generará una notificación advirtiendo de ello.

Otra funcionalidad del protocolo es el retain message o mensaje retenido. EL bróker tiene guardado un mensaje retain message en temperatures con el nombre de dispositivo ESP32A. Por tanto, cada vez que la plataforma de gestión o el terminal móvil se suscriban a temperatures, el bróker les enviará un mensaje. De esta manera el dispositivo ESP32 no tiene que mandar el nombre de su dispositivo cada vez que manda la temperatura (supuestamente hay más dispositivos ESP32 y es la manera de diferenciarlos). De esta manera la plataforma de gestión cuando realiza el guardado de datos también incluye el nombre ESP32A junto con la temperatura en la base de datos.

Cada vez que se inicia la aplicación móvil se manda un retain message con la fecha del día. De esta manera cuando la plataforma de gestión se suscribe a el tópico location obtiene ese mensaje y lo usa para guardarlo junto con los datos de localización que recibe y la hora.

Este sistema IoT está diseñado de la tal manera que pueda servir como referente a la hora de realizar un sistema IoT usando el protocolo MQTT. En un sistema IoT hay muchos dispositivos obteniendo datos de su entorno y publicando la información al sistema. En el sistema suele haber una plataforma que se encarga de procesar estos datos (en nuestro caso la platafroma de gestión), guardarlo, y enviar respuestas.

Las respuestas pueden ser datos que mostrar a los usuarios del sistema, el teléfono móvil informa de alertas mediante notificaciones. Pero también tener el objetivo de realizar una actuación. En nuestro caso la actuación es encender un led, pero podría ser que el ESP32 accionara el aire acondicionado debido a las temperaturas elevadas obtenidas.

Aclaración

Que el ESP32 mida la temperatura, la envié a la plataforma de gestión, sea comprobada y vuelta a enviar al mismo ESP32 no tiene mucho sentido, ya que ese proceso lo podría realizar el ESP32 por si solo. Pero este proceso es realizado a modo de concepto, que es el siguiente:

El ESP32 mide la temperatura, es enviada a la plataforma de gestión, y esta la analiza. En el caso de disponer de más dispositivos ESP32 y sensores de temperatura el análisis podría ser realizar una media y si esta es superior a 30 grados, enviar la temperatura

media de vuelta al ESP32. Como no disponemos de los dispositivos en su defecto se compara la temperatura con un valor determinado.

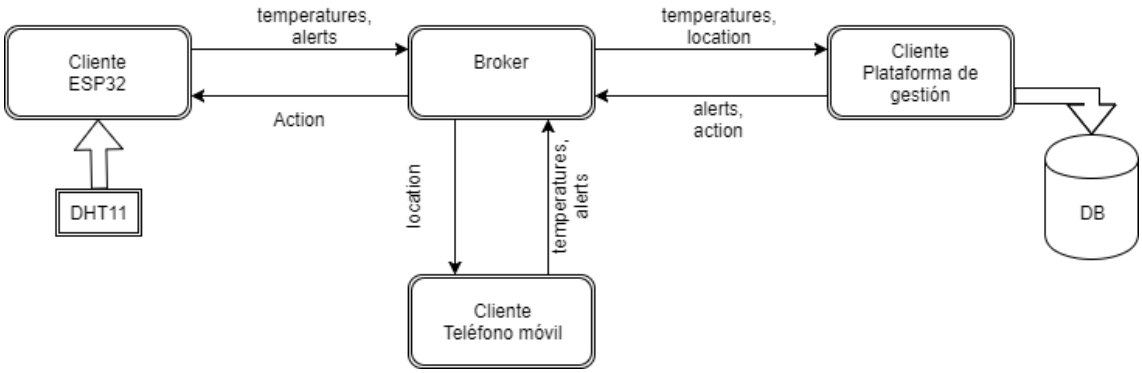


Figura (3)

3.2. Arquitectura del sistema

Una vez descrito el funcionamiento del sistema se procede a la descripción de la arquitectura del sistema, que incluye hardware, sistemas operativos, software que utilizan los diferentes clientes del sistema y la arquitectura de la red.

Cliente ESP32

El cliente ESP32 está formado por el microcontrolador ESP32 y un sensor DHT11 que mide la temperatura.

El ESP32 es un microcontrolador que está alimentado mediante conexión micro usb, en el que se han de destacar algunas características relevantes para el sistema:

- Comunicación al sistema se realizará mediante WiFi:
  - WiFi :802.11 b/g/n (802.11n up to 150 Mbps) en frecuencias 2.4 ~ 2.5 GHz
- Hardware:
  - GPIO, sensor de contacto capacitivo, ADC, DAC
  - Voltaje operativo: 2.7 - 3.6V

El sensor DHT11 que presenta algunas características relevantes para el sistema como son:

Tabla(2)

	Conditions	Minimum	Typical	Maximum
Power Supply	DC	3V	5V	5.5V
Current Supply	Measuring	0.5mA		2.5mA
	Average	0.2mA		1mA
	Standby	100uA		150uA
Sampling period	Second	1		

Note: Sampling period at intervals should be no less than 1 second.

Un detalle importante que podemos ver en la tabla es que la velocidad máxima a la que puede obtener información de temperatura el sensor dht11 es cada 1 segundo. Por lo que esto nos genera una limitación en el sistema en cuanto a la velocidad a la cual se puede transmitir desde el cliente situado en el ESP32.

El sensor DHT11 consta de 3 conexiones, una para alimentarse, una para conectar a tierra y la tercera por la cual transmite los datos de temperatura. Hace falta recalcar que el sensor es un sensor digital por lo que todos los GPIO de entrada de datos servirán. Si fuese analógico condicionaría el GPIO usado ya que no todos cuentan con conversor analógico digital. En este caso se ha conectado al GPIO26

El dispositivo ESP32 cuenta con un PIN que genera una tensión de 3,3V y otro que de tierra por lo que el sensor DHT11 se conectar al ESP32 para alimentarse.

En la siguiente figura se muestra la conexión entre el ESP32 y el DHT11.

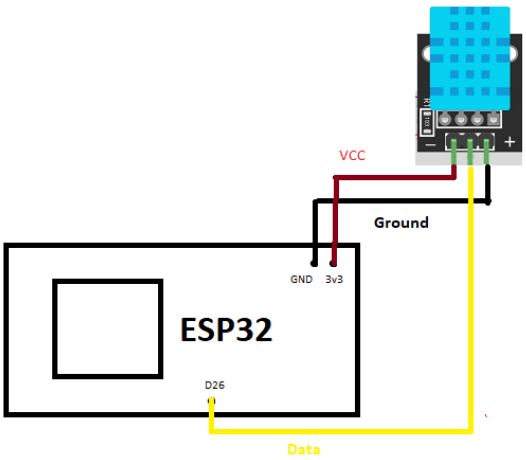


Figura (4)

En cuanto al software el ESP32 utiliza arduino. Este software incluye un entorno de desarrollo (IDE) que implemente el lenguaje de programacíon de arduino y un bootloader para ejecutar la placa. Se ha utilizado el software the arduino por su sencillez y facilidad de uso por la cual razón es uno de los lenguajes más utilizados en cuento a dispositivos ligeros con especificaciones recortadas. Al ser uno de los más usados cuenta con un gran abanico de librerías y API's de terceros para poder realizar llevar a cabo funciones o tareas que no son propias del software. En nuestro caso para poder comunicarse mediante el protocolo MQTT y obtener datos del sensor DHT11. Las librerías usadas, métodos y funciones estarán explicadas posteriormente.

Dispositivo móvil

El terminal usado es un Xiaomi redmi 4 Pro que consta de un procesador a frecuencia 2GHz y con 3GB de memora RAM. Transmite los mensajes al sistema mediante la tecnología 4G. Dispone de localizador GPS usado para la obtención de localización.

El terminal móvil tiene Android como sistema operativo y para desarrollar la aplicación se ha usado el entorno de desarrollo integrado Android Studio. Android es el sistema operativo más utilizado a nivel de terminales móviles por lo que permite que más usuarios puedan implementar la aplicación en su Smartphone. Es un SO libre por lo que facilita la construcción de librerías, API's y Kits de desarrollo.

Plataforma de gestión:

La plataforma de gestión esta implementada en un ordenador portátil con un procesador i7 a frecuencia 2.50Ghz 2.59Gz, con 8 GB de RAM y con sistema operativo Ubuntu 16.04 LTS de 64 bits. Envía los mensajes al bróker mediante WiFi con un ancho de banda de aprox. 28MB/s casi simétricos. Estos valores no son estables ya que está conectada a la red EDUROAM con muchos usuarios y diferentes cargas de trabajo.

El programa que ejecuta la base de datos esta realizado en Python (versión) que ya venía implementado en el sistema operativo.

Bróker

El bróker esta implementado en un ordenador de sobremesa con un procesador Intel core 2 CPU 6320 con una frecuencia de 1.86 Ghz x2, 4 GB RAM y el Ubuntu 16.04 LTS de 64bits. Dispone de una conexión ethernet a la red y utiliza el mosquitto como software.

Mosquitto es un bróker open source que implementa el protocolo MQTT versión 3.1 y 3.1.1 para gestionar el intercambio de mensajes. Todos los clientes que conectan al bróker lo hacen mediante la versión 3.1.1

Arquitectura de la red

A la hora de diseñar la estructura de la red es imprescindible que cualquier cliente pueda acceder al bróker que es el elemento central donde se dirigen todos los mensajes. El bróker está situado en un laboratorio de la Universidad Politécnica de Cataluña (UPC) y está configurado con una IP pública. Gracias a esta IP pública, cualquier dispositivo conectado a internet puede intercambiar mensajes con el bróker.

Como se ha comentado anteriormente, el dispositivo ESP32 se comunica mediante WiFi. El microcontrolador se conecta a un router WiFi que está en laboratorio de la UPC. Al estar en la misma red que el bróker, aunque los paquetes vayan destinados a la dirección pública no salen de la red interna de la UPC.

La plataforma de gestión también está conectada a la red interna de la UPC mediante WiFi, usando EDUROAM. Por tanto, sucede lo mismo que con el ESP32, que los paquetes no llegan a salir de la red privada de la UPC.

Por ultimo tenemos el terminal móvil, que se comunica mediante 4G, por lo que los mensajes sí que viajan a través de internet y gracias a que el bróker tiene una IP publica llegan sin tener que realizar ninguna configuración de red extra.

La arquitectura de la red se ilustra en la siguiente figura:

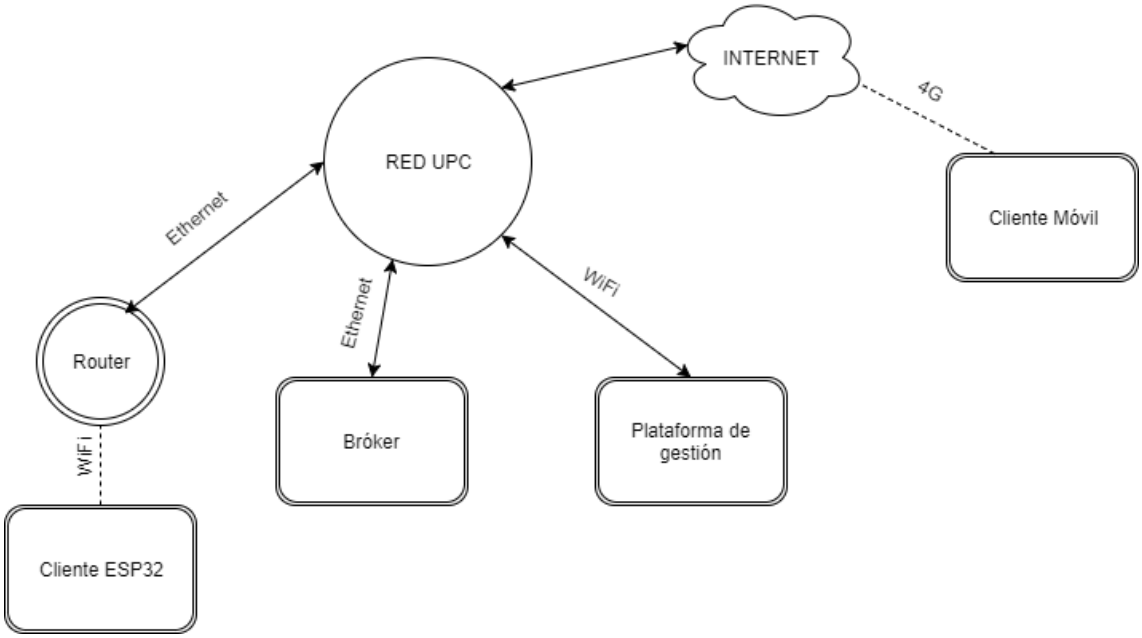


Figura (5)

3.3. Tópicos y QoS

En el protocolo MQTT se define la comunicación de los menajes a través de los tópicos en los cuales los clientes podrán publicar mensajes o recibir aquellos mensajes que les interesen. Por tanto, hay que definir estos tópicos que el bróker usará para dirigir los mensajes hacía los clientes suscritos a ellos. Los tópicos en el protocolo MQTT son Strings UTF8.

Una vez tenemos la descripción funcional del sistema se pueden definir los publicadores, suscriptores, los tópicos y las QoS asociados a ellos.

Cliente ESP32:

- Publicador a tópicos temperatures (QoS0) y alerts (QoS0)
- Suscritor a tópico action (QoS1)

Cliente Móvil:

- Publicador a tópico location (QoS1)
- Suscritor a tópicos alerts (QoS2) y temperatures (QoS0)

Cliente plataforma de gestión:

- Publicador a tópico action (QoS1) y alerts (Qos2)
- Suscritor a tópicos temperatures (QoS0) y location (QoS1)

En el apartado de antecedente hablábamos de la capacidad que tienen los tópicos para realizar distinciones de niveles. En nuestro sistema no tenemos una configuración idónea

para realizar esto, ya que tenemos pocos dispositivos y pocos tópicos y no obtendríamos mucho beneficio.

Si analizáramos la comunicación solamente a nivel de tópicos tendríamos el resultado de la figura de abajo.

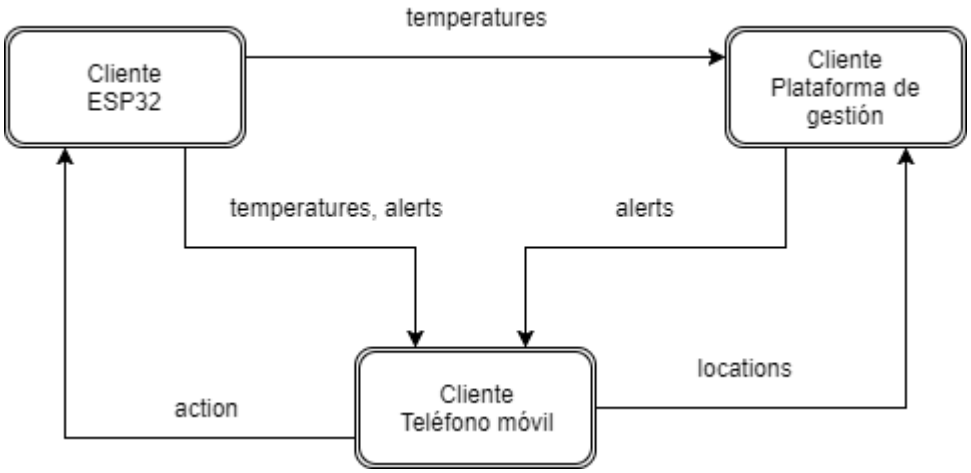


Figura (6)

3.4. Implementación del sistema

3.4.1. Bróker Ubuntu

Como se comenta en la introducción el bróker es el elemento central del sistema encargado de distribuir los mensajes entre los diferentes clientes y vamos a usar el software de mosquitto. Mosquitto es un bróker open source que implementa el protocolo MQTT versión 3.1 y 3.1.1 para gestionar el intercambio de mensajes.

En el sistema se añade un elemento de seguridad que es el de autentificación, por el cual un dispositivo necesita un usuario y contraseña para conectarse al bróker.

El primer paso es crear un fichero con los usuarios y contraseñas para que mosquitto tenga un registro de ellos. En la propia instalación de mosquitto nos viene un comando para crear dicho fichero con las parejas usuario contraseña, mosquitto_passwd con la opción de crear –c, con el nombre del archivo, el usuario y una vez ejecutado te pedirá que introduzcas la contraseña del usuario. Estos tipos de comandos se han de realizar con permisos de administrador.

```
sudo mosquitto_passwd -c authentication esp
```

Si se quieren añadir más usuarios y contraseñas es el mismo comando, pero en vez de –c sustituido por –b y añadiendo la contraseña al final del comando.

```
sudo mosquitto_passwd -b authentication xiaomi pwxiaomi
```

En el fichero se guarda de forma pareja “usuario:contraseña”, pero al crearse la contraseña esta es directamente encriptada en un formato similar al crypt(3), que es una librería de c para generar ficheros de contraseñas. Por lo cual no se puede tener acceso fácil a esa contraseña.

Una vez ya tenemos el fichero con los usuarios y sus contraseñas hay que modificar el fichero de configuración llamado mosquitto.conf ubicada en /etc/mosquitto .

En ella se prohíbe la conexión de clientes sin autentificación y se le indica la ruta hasta donde se encuentra el fichero con los usuarios y contraseñas para que mosquitto pueda acceder a ellos.

```
allow_anonymous false
```

```
password_file /etc/mosquitto/authentication
```

Los cambios no entrarán en efecto hasta que no se reinicie el servicio de mosquitto, ya que la configuración se carga al iniciar el servicio. Por tanto, usaremos el mismo comando una vez habíamos acabado la instalación para reiniciarlo.

Cabe decir que este nivel de seguridad es muy básico ya que a la hora de que los usuarios se conecten al bróker enviarán el usuario y la contraseña visible por lo que podría obtenido con la ayuda de analizadores de red.

Mosquitto implementa ssl/tls en su sistema por lo que en caso necesario podrán emplearse para realizar una comunicación segura. Pero en este demostrador no se considera necesario ya que como comentábamos es un trabajo base para que sirva para terceras personas y en ese caso ya valorarán si necesitan el plus de seguridad en su sistema o no.

3.4.2. Bróker Raspbertty

En este proyecto se quiere dar una versión alternativa a la instalación del bróker en un ordenador de sobremesa, con la que demostrar esa gran versatilidad del protocolo MQTT para poder ser aplicado en una gran variedad de tecnologías. Esta una versión más económica para realizar el proyecto, por la que nuestro bróker tendrá menos capacidad de procesamiento al tratarse de una raspberry pi 2, con sistema operativo RaspbainTambién se quiere mostrar cómo implementar el bróker si no se dispone de una ip pública en tu red privada.

Configuración red

Como hemos dicho este bróker se encuentra en una red privada y no dispone de una dirección ip publica con la que elementos externos puedan acceder a ella sin realizar una configuración de red.

El primer paso es averiguar la IP pública que nuestro servicio de internet tiene asociada para que lleguen los paquetes hasta nuestro router. Esta será la necesaria para usarse como end point en nuestro sistema. En un entorno Industria 4.0 el administrador de la red gestiona la Ip pública, en un entorno doméstico podemos realizar un procedimiento sencillo en el que solamente tecleando en google “¿Cuál es mi ip publica?” te la reconocen. De esta forma los mensajes ya podrán acceder al router que hace de enlace entre internet y la red privada.

Un detalle a tener en cuenta es que la dirección ip publica del router que te ofrece el proveedor de servicios de internet es dinámica, por lo cual puede variar. Normalmente esto sucede cuando el router se desconecta. Por tanto, la solución que se plantea en este apartado es de cara a realizar un sistema a nivel de usuario particular, ya que no tendría sentido a nivel de sistema estable para usarse en empresa. Pero como se indicaba en la

introducción este demostrador quiere poder dar solución a todo tipo de personas que quieran aprender sobre el protocolo MQTT y puedan usar este documento como referencia.

Posteriormente se han de tener en cuenta 2 cosas; en una red local, el router es el encargado de dar ip privadas mediante el servicio de dhcp automático, por lo que puede que la ip privada varíe al arrancar el dispositivo, y la segunda es el enrutado interno de la red para llevar esos paquetes hasta la raspberry pi2.

La primera configuración del router que se ha de llevar a cabo es la de desactivar el dhcp automático para nuestra raspberry pi y establecer la opción manual. En esta configuración manual se ha de asignar una ip privada dentro del rango de nuestra red a la dirección MAC de nuestra raspberry pi. De esta manera, al iniciar la raspberry, detecta la MAC y le asigna siempre esa ip.

Una vez ya está definida la ip privada, se ha de dirigir los paquetes entrantes de nuestro sistema a esa Ip y se hace mediante el filtrado de paquetes. En el router te deja configurar varias reglas de filtrado y en nuestro caso se ha de crear una regla para que los paquetes entrantes dirigidos al puerto 1883 (que es el del MQTT) se dirijan a la ip privada asignada anteriormente.

3.4.3. Cliente ESP32

Librerías:

El código del esp32 requiere librerías necesarias para conectar y comunicarse con el bróker. Primero de todo, la comunicación se hará a través de la tarjeta WiFi que dispone el dispositivo. Por esta razón nuestro código debe importar esta librería WiFi.h. Esta librería ya está incluida en los archivos que descargamos para la instalación, que se adjunta en el anexo.

La siguiente parte que entra en juego a la hora de transmitir es el protocolo MQTT. En este caso vamos a usar una librería cliente denominada PubSubClient. Esta librería nos permite llevar a cabo nuestro cometido en cuanto a conexión con el bróker, publicar y recibir mensajes.

Esta librería trabaja con mensajes de publicación con calidad de servicio 0. Se puede suscribir a tópicos con QoS 0 o 1. Esto ya nos es conveniente porque no interesa que un dispositivo con poco procesado y que manda temperaturas cada 2 segundos tenga que utilizar el mecanismo de reconocimiento de mensajes todo el rato. Si se pierde un mensaje de temperatura no es signiticativo.

Si alguien está interesado en reproducir este proyecto y en su caso los datos no se pueden permitir la opción de ser perdidos, esta no es la librería que debería escoger.

Esta librería permite trabajar con mensajes retain a la hora de publicar mensajes y especificar un will message al conectar, dos funcionalidades útiles que presenta el protocolo MQTT.

También será necesaria la librería DHT, que es la que nos aporta los métodos necesarios para obtener la temperatura del sensor externo DHT11.

Código y desarrollo

En este apartado se mostrarán los métodos más relevantes junto con sus descripciones, funciones y atributos necesarios, sin llegar a exponer todo el código. El código se adjuntará en el anexo para poder ser utilizado para reproducir el sistema y constará de comentarios explicativos.

El lenguaje de programación de arduino se caracteriza por tener 3 partes diferencias. Una primera parte donde se llevan a cabo las inclusiones de librerías, definiciones y se pueden incluir constantes, variables y funciones. Un setup que se inicia al arrancar el dispositivo y solo se ejecutará una vez. Por último, el loop donde se ejecuta el código de manera recurrente en bucle.

Para que el programa pueda recibir mensajes, lo primero que se tiene que hacer es definir en la función en la que especificar las acciones al recibir un mensaje de los tópicos suscritos. En este caso se enciende el led de la placa durante 1 segundo.

```
void callback(char* topic, byte* payload, unsigned int length) {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
}
```

setup

Los primeros métodos a destacar dentro del setup son los que hacen referencia a la conectividad Wifi. Encontramos 2 metodos; el begin que inicia la conexión wifi y provee del estado actual de la conexión, y el status que permite comprobar el estado en el que se encuentra la conexión.

En nuestro caso para hacer iniciar la conexión a la red wifi deseada usaremos el método en el que se le pasan 2 atributos, el nombre de la red y su contraseña.

```
WiFi.begin(ssid, password);
```

La función status la usaremos dentro de un bucle while como condición para que el programa no avance hasta que el estado sea conectado. El estado conectado se define dentro de la librería como WL_CONNECTED.

```
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi");
  }
```

Una vez ya se ha establecido la conexión wifi toca hacer lo mismo con la conexión al bróker. Para ello el primer paso es dotar de las credenciales del servidor al cliente que son; la dirección ip (o url) y el puerto.

```
clientMQTT.setServer(mqttServer, mqttPort);
```

Al cliente también se le establece un callback en el cual se le hace referencia a la función anteriormente definida.

```
clientMQTT.setCallback(callback); // callback for the client to recived message
```

Una vez se le han especificado los datos del servidor al cliente, toca hacer la conexión. Esta conexión se ha de llevar a cabo dentro de un bucle while en el cual se mantendrá hasta que el programa detecte que se ha conectado. Para hacer esta comprobación la librería pubSubClient incorpora un método `connected()` que devuelve cierto o falso.

El método que conecta con el servidor es el expresado aquí debajo y al cual se le han de especificar ciertos parámetros; un identificador de cliente, el usuario y pasword definidos en la configuración el bróker. En el sistema también usamos una de las funcionalidades que tiene el protocolo MQTT que es el will message. Por tanto, también se le han de especificar a la hora de hacer la conexión con el servidor; will topic, la calidad de servicio de ese will tópic, si el flag de retain está activo de ese will message y finalmente el contenido del este.

```
clientMQTT.connect("Esp32Nuevo", mqttUser, mqttPassword, willTopic, willQoS, willRetain, willMessage)
```

Este método devuelve true o false indicando si se ha podido hacer la conexión o no, y en caso de que se conecte podemos realizar la suscripción al tópico action, con QoS1.

```
clientMQTT.subscribe("action", 1);
```

Y por último dentro del setup, tenemos que iniciar el sensor con la sentencia begin similar a la hecho con la wifi pero sin parámetros `dht.begin();`

Loop

En el loop se incluye el código que va a ser ejecutado de manera recurrente, ya que esta es la manera en la que trabajan los dispositivos programados con arduino IDL.

Primeramente se ha de obtener la temperatura mediante un sensor, y eso se hace mediante el método `dht.readTemperature()` asignado a la instancia que teníamos del sensor.

Una vez se obtiene la temperatura ya se puede publicar al sistema mediante el método publish en el que se le especifica el tópico y la carga que corresponde a la temperatura.

```
clientMQTT.publish("temperatures", payload)
```

El cliente del ESP32 también comprueba si se ha apretado el botón con una función que pertenecen a la librería que instalamos propia del ESP32, ya que el botón que se comprueba está integrado en el chip. Este método puede devolver LOW or HIGH para indicar si se ha apretado o no el botón.

```
digitalRead(BUTTON_PIN)
```

Y en caso de ser apretado se publicaría otro mensaje utilizando el mismo método publish de antes, pero con tópico alerts para informar al sistema que se ha apretado el botón.

Para que el cliente pueda procesar los mensajes recibidos, se ha de llamar a la función loop, de forma periódica.

```
clientMQTT.loop();
```

Para acabar codigo se llama a la función delay para que nuestro sistema no vuelva a empezar el código del loop, sino que espere 2 segundos.

3.4.4. Cliente terminal Móvil

Este proyecto se ha realizado medite la plataforma de desarrollo andriod studio. Por lo que lo primero que tenemos que hacer es instalarlo. Se puede realizar la instalación tanto windows como linux. Para windows solamente se tiene que acceder a la página oficial y descargar el ejecutable. Hace falta destacar que el código es bastante extenso, pero muchas de las líneas de código son para el funcionamiento de la aplicación, creación de la interfaz de usuario y programación básica de android, por lo que en esta parte solo se expondrán los métodos, atributos y conceptos necesarios para la obtención de los datos de localización, comunicación con el bróker MQTT y permisos, estos últimos ya que son requeridos tanto para uno como para otro.

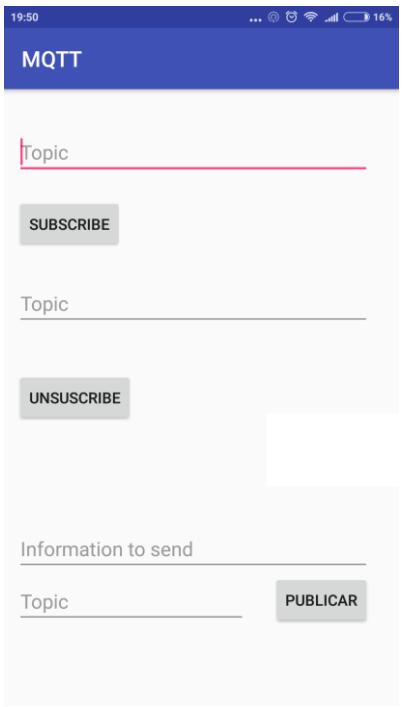


Figura (7)

Una vez instalado, lo primero que se ha de hacer es incluir las librerías necesarias para poder trabajar con ellas. Como hemos dicho el dispositivo móvil obtiene la localización y para ello recurriremos a métodos y funciones de google play services. Por esta razón se ha de incluir la SDK (sofware developer kit) de google play services. Esta se puede descargar desde el mismo android studio desde: tools/android/sdk Manager/andorid sdk/sdk tools/ google play services

Una vez realizado este paso, toca incluirlas en la aplicación que estamos creando. Para ello nos dirigimos dentro de la carpeta de app al build.gralde y añadimos las dependencias de localización de google play services. Las demás dependencias de google play services no son necesarias incluirlas.

```
compile 'com.google.android.gms:play-services-location:7.0.0'
```

Las otras librerías necesarias son las que respectan a la comunicación mediante el protocolo MQTT. Se han de descargar las librerías org.eclipse.paho.android.service-1.1.1.jar y org.eclipse.paho.client.mqttv3-1.1.1.jar. Una vez descargadas se han de incluir en la carpeta libs del proyecto creado. Esta carpeta está dentro de la carpeta app de nuestra aplicación. En caso de no estar se ha de crear. Una vez ya cargadas en el programa se ha de añadir las dependencias igual que en el caso anterior:

```
compile files('libs/org.eclipse.paho.android.service-1.1.1.jar')
compile files('libs/org.eclipse.paho.client.mqttv3-1.1.1.jar')
```

En el sistema android hay ciertas funciones o métodos que requieren permisos para poder ejecutarse. En este proyecto para la comunicación MQTT se deben añadir las siguientes librerías en el AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
android:name="android.permission.READ_PHONE_STATE" />
```

También se ha de crear enlace con Paho Android Services añadiendo la siguiente sentencia:

```
<service
android:name="org.eclipse.paho.android.service.MqttService" />
```

Por último, para acceder a la localización del dispositivo también se requieren permisos, hay 2 permisos uno de alta definición de la localización y otra que es menor, en este proyecto se incluye la de mayor precisión.

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
```

En el código desarrollado para la aplicación móvil existen 2 partes que se encargan cada uno de una funcionalidad, por un lado, el obtener datos de localización, y por otra la parte de comunicación del protocolo MQTT. Tanto el google play services como la librería de MQTT realizan conexiones por lo cual implementan callbacks para estar escuchando las respuestas a esas conexiones e intercambios de mensajes ya sea con el bróker o el servicio de google play y han de ser definidos en la mainActivity que en este caso se ha nombrado como MQTT_Transmissio mediante implements. También es necesario definir que la MQTT implementa el locationListener para que la ampliación puede realizar acciones cuando el sistema detecta cambios de localización.

Locations:

Para obtener las localizaciones primero se ha de crear una instancia a la API de cliente de google play services dentro del onCreate(), donde se ejecutan los componentes fundamentales de la actividad. Para construir el cliente del google service se especifica la API requerida de las localizaciones, un connection callbak listener para recibir los eventos de las conexiones y el listener para las conexiones fallidas.

```
mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
```

```
        .addOnConnectionFailedListener(this)
        .addApi(LocationServices.API)
        .build();
```

También debemos añadir en el onCreate el método connect() para conectar el cliente con la google play services una vez se inicia la apliación.

El siguiente paso es definir los intervalos para la obtención de la localización. Esto se hace mediante la clase locationRequest. Esta clase nos permite realizar algunas configuraciones en cuanto a las localizaciones requeridas mediante métodos.

El metodo setPriotity permite definir la precisión de las localizaciones obtenidas. El método setInterval establece el intervalo, en milisegundos, entre peticiones de localización. Hace falta recalcar que si otras aplicaciones el método setInterval es inexacto, y por ello no recibirá la actualización de localización exactamente en ese instante. Podría ser que recibiera la localización antes, si otra aplicación requiere una localización, que no la recibiera, en caso de que no estuviera disponible para el teléfono en ese instante, o que la recibiera algo después. Por esta razón es importante definir un intervalo mínimo entre actualizaciones de localización para evitar que si otra aplicación está requiriendo muy seguidamente la localización, esto afecte en gran medida a los tiempos diseñados para nuestra aplicación, eso se realiza con el método setFastestInterval.

Una vez definidos los parámetros de las peticiones de localización tenemos que requerir estas localizaciones y se hace mediante el metodo requestLocationUpdates al que se le pasa el cliente de google que hemos creado en el metodo onCreate, el location request creado y un pending intent.

La configuración del location request y la petición de actualizaciones se definirán dentro del método startLocationUpdates para poder llamarlo desde el método onConnected. Este método responde a los callbacks del cliente de google que hemos creado y en el que sobrescribiremos para que una vez conectado llame a el método startLocationUpdates.

```
protected void startLocationUpdates() {
    // Create the location request
    mLocationRequest = LocationRequest.create()
            .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
            .setInterval(UPDATE_INTERVAL) // 10 segundos
            .setFastestInterval(FASTEST_INTERVAL); //9 segundos
    LocationServices.FusedLocationApi.requestLocationUpdates(mGoog
leApiClient, mLocationRequest, this);
}
@Override
public void onConnected(Bundle bundle) { //ones the connection to
google services it's done
    startLocationUpdates();
}
```

En el sistema expuesto en este documento el intervalo que establecemos es de 10 segundos. Si una persona camina a 5km/h si realizamos la petición cada 10 segundos las distancias serán de 13 metros de separación. El intervalo de tiempo es un valor debe ser modificado dependiendo de la utilidad o posible funcionalidad que alguien le quiera dar. El que esta no sea exactamente a los 10 segundos no tiene repercusión en el sistema puesto que la información de localización no es determinante si viene un segundo antes o después.

Para finalizar con la localización falta definir qué hacer cuando hay una actualización de la posición. Para ello tenemos que sobrescribir método onLocationchanged. En él se pueden extraer la latitud y la longitud de la localización con los métodos getLatitude() getLongitude(). También obtenemos la hora minuto y segundo del día en la que se ha obtenido esta localización. Esto puede ser muy útil para poder calcular luego velocidades o conocer donde se encontraba y en qué momento el dispositivo mirando los datos en la base de datos incluida en la plataforma de gestión.

En este método es se enlazan las 2 partes de la aplicación, la de obtener localizaciones junto con la comunicación con el sistema. En este método se llama al método para publicar el mensaje con la localización al sistema mediante el tópico location.

```
@Override
public void onLocationChanged(Location location) {
    DateFormat df = new SimpleDateFormat("HH:mm:ss");
    Date today = Calendar.getInstance().getTime();
    String actualTime = df.format(today);
    String msgGPS = Double.toString(location.getLatitude()) + ","
+
            Double.toString(location.getLongitude()) + ","
            + actualTime;
    try {
        publishMessage(mqttAndroidClient,      msgGPS,      qosGps,
topicGPS);
    } catch (MqttException e) {
        e.printStackTrace();
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}
```

MQTT comunication:

Cuando arranca la aplicación y ejecuta el método onCreate, ahí es donde se han de realizar las conexiones con el bróker y las suscripciones a los tópicos pertinentes. Estas acciones se van a definir dentro de un método llamado startMqttClient al cual se le pasa

como parámetros el contexto de la aplicación, la dirección del bróker y la identificación del cliente y como resultado devuelve un MqttAndroidClient

La primera acción del método es crear este cliente MQTT con el constructor asignándole, los mismos parámetros que recibe la clase en la cual está incluido. Una vez tenemos el cliente ya podemos realizar la conexión con el bróker con el método connect. El método connect recibe como parámetro un objecto MqttConnectOption que es el que permite establecer las opciones de la conexión MQTT con el bróker. Estas opciones están definidas en un método llamado MQTTConnectionOption(explicado a posteriori).

Connect devuelve un ImqttToken que nos permite luego mediante el método setActionCallback crear un ImqttActionListener que informará de si la conexión se ha realizado con éxito o no. También permite sobrescribir sus métodos onSuccess y onFailure para realizar acciones dependiendo del resultado de la conexión.

Cuando la conexión se ha establecido con éxito es el momento para enviar el mensaje de suscripción al bróker con los tópicos y calidades de servicio que deseamos. El método subscribe requiere como parámetros los tópicos a los cuales se quiere subscribir y las respectivas cualidades de servicio. El método subscribe es similar al connect en cuanto a que devuelve un ImqttToken, por lo que también podemos asignar funcionalidades para cuando la respuesta suscripción es positiva o negativa.

```
public MqttAndroidClient startMqttClient(Context context, String brokerUrl, String clientId) {
    mqttAndroidClient = new MqttAndroidClient(context, brokerUrl, clientId);
    try {
        IMqttToken token = mqttAndroidClient.connect(MQTTConnectionOption());
        token.setActionCallback(new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                mqttAndroidClient.setCallback(MQTT_Transmission.this);
                try {
                    DateFormat df = new SimpleDateFormat("dd/MM/yyyy");
                    Date today = Calendar.getInstance().getTime();
                    String todayString = df.format(today);
                    publishRetainMessage(mqttAndroidClient,"Date_"+todayString,topicGPS);
                    final IMqttToken subToken = mqttAndroidClient.subscribe(subTopics,qoss);
                    subToken.setActionCallback(new IMqttActionListener() {
```

```
                        @Override
                        public void onSuccess(IMqttToken asyncActionToken) {
                            Context context = getApplicationContext();
                            int duration = Toast.LENGTH_SHORT;
                            Toast.makeText(context, "Subscribed to: " + subTopics[0] + " & " + subTopics[1], duration).show();
                        }
                        @Override
                        public void onFailure(IMqttToken asyncActionToken,
                                              Throwable exception) {
                            Context context = getApplicationContext();
                            int duration = Toast.LENGTH_SHORT;
                            Toast.makeText(context, "Impossible to subscribe", duration).show();
                        }
                    });
                } catch (MqttException e) {
                    e.printStackTrace();
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }
            }
            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
                Context context = getApplicationContext();
                int duration = Toast.LENGTH_SHORT;
                Toast.makeText(context, "Impossible to connect with broker", duration).show();
            }
        });
    } catch (MqttException e) {
```

```
            e.printStackTrace();
        }
        return mqttAndroidClient;
    }
```

Retomando la configuración de conexión, lo primero que se ha de especificar en ella son las credenciales mediante setUserName y setPassword. También definimos el intervalo máximo sin enviar mensajes al bróker de 30 segundos ya que por defecto son 60, siendo este, 6 veces superior al proceso normal de envió de localización al bróker (cada 10 segundos). También se especifica que el servidor y el cliente han de mantener el estado de sesión de la conexión, mediante el setCleanSessin(false).

Este mantenimiento de sesión no tendría mucho sentido sin la activación el método setAutomaticReconnect por el cual el cliente intentará conectarse en caso de pérdida de conexión con el bróker.

```
private MqttConnectOptions MQTTConnectionOption() {
    MqttConnectOptions mqttConnectOptions = new
MqttConnectOptions();
    mqttConnectOptions.setCleanSession(false);
    mqttConnectOptions.setAutomaticReconnect(true);
    mqttConnectOptions.setKeepAliveInterval(20);
    mqttConnectOptions.setUserName(mqttUserName);
    mqttConnectOptions.setPassword(mqttPwd.toCharArray());
    return mqttConnectOptions;
}
```

Nuestro sistema IoT está diseñado para que no se tenga que enviar la fecha continuamente a la plataforma de gestión. El flag retain que se puede activar al enviar un mensaje, este haciendo que el bróker guarde ese mensaje y lo publique como primer mensaje a los demás clientes que se subscriben a él. Por esta razón la aplicación ha de mandar ese mensaje cada vez que se conecta para que así el servidor pueda actualizar la fecha si esta ha cambiado. Por lo que en el onSucces del método connect hemos de enviar un mensaje con el flag retain activo al tópico location y para ello se crea una función llamada publishRetainMessage que incluye el método publish de la librería MQTT junto con algunas configuraciones que se le aplican a dicho mensjae. Este método es igual al de publishMessage visto en el apartado de localizaciones para enviarla al sistema siendo la única diferencia el retain flag que está activo para la función PublishRetainMessage. Los 2 métodos requieren como parámetros un cllienteMQTT, el mensaje y el tópico.

```
public void publishMessage(@NonNull MqttAndroidClient client,
        @NonNull String message, int qos, @NonNull String topic)
        throws MqttException, UnsupportedEncodingException {
```

```
    byte[] encodedPayload = message.getBytes("UTF-8");
    MqttMessage mqttMessage = new MqttMessage(encodedPayload);
mqttMessage.setRetained(false); //El método publishRetainMessage estaría en true
    mqttMessage.setQos(qos);
    client.publish(topic, mqttMessage);
}
```

El siguiente punto de la applicación es la parte de recepción de mensajes y esto es realizado en el método messageArribed. Este método se llama cuando llega un mensaje a la aplicación de uno de los tópicos subscritos y tiene como parámetros el tópico del mensaje recibido y mes mensaje en formato MqttMessage.

En este método lo primordial es distinguir los tópicos de los mensajes haciendo comparación del tópico del mensaje recibido con el de los que estamos suscritos y entonces dependiendo de cuál, hacer una acción u otra. En nuestro caso si el mensaje recibido es del tópico temperaturas lo mostramos con un toast y si es una alerta el sistema generará una notificación que procesará el sistema operativo.

La aplicación también cuenta con funcionalidades para poder suscribirte a tópicos, publicar mensajes a tópicos y des-suscribirse permitiendo al usuario realizar estas interacciones a través te textos y botones. La realización de estas funcionalidades ya se han comentado en los párrafos anteriores, pero queda el método unsusbcribe que es muy similar los métodos connect y subscribe. Devuelve un toquen y espera respuesta afirmativa o negativa, y este método solo necesita como parámetro el tópico al cual des-suscribirse.

3.4.5. Cliente plataforma gestión

El sistema operativo ubuntu ya incluye un compilador python por lo que de cara al programa solo necesitaremos preocuparnos por la instalación de las librerías necesarias para la comunicación del protocolo MQTT. Es librería es la paho-mqtt y para instalarla la haremos mediante los comandos:

```
pip install paho-mqtt
```

Para obtener todo el código y algunos ejemplos se ha de clonar el repositorio con git.

```
git clone git://git.eclipse.org/gitroot/paho/org.eclipse.paho.mqtt.python.git
```

Una vez ya tenemos el repositorio cargado toca instalarlo

```
cd org.eclipse.paho.mqtt.python
python setup.py install
```

Esto con lo que respecta a python, pero la plataforma de gestión también incluye una base de datos MySQL.

1) Para realizar la instalación deberemos lanzar el comando en el terminal. Con este comando instalara la base de datos y otros paquetes necesarios. Durante la instalación nos requerirá de una contraseña para acceder a dicha base de datos.

```
sudo apt-get install mysql-server
```

2) Posteriormente obtenemos todos los paquetes del módulo MySQLdb. MySQLdb nos permitirá actuar con esta base de MySQL de forma más sencilla desde python.

```
apt-cache search MySQLdb
```

3)Procedemos a la instalación de MySQLdb con el comando.

```
sudo apt-get install python-mysqldb
```

Configuración:

En el apartado de configuración se realizan las tareas necesarias para la creación de la base de datos necesaria para guardar las temperaturas y las localizaciones.

1) Con el cliente mysql vamos a crear un usuario y una contraseña para acceder a la base de datos.

```
mysql -u root -p  // Nos conectamos como usuario root y asignamos unacontraseña // contraseña Pelota5
```

2) Ahora toca crear nuestra base de datos con el siguiente comando.

```
mysql> CREATE DATABASE locationDB
```

3) El siguiente paso es crear un usuario para poder acceder a esta base de datos y que se requerirá posteriormente desde el programa de python.

```
Mysql> CREATE USER 'adminDB'@'localhost' IDENTIFIED BY 'adminDb1';
```

4)Con el comando USE, se define la base de datos que queramos.

```
mysql> USE locationdb;
```

5) Una vez creado el usuario el siguiente paso es darle permisos para que pueda acceder a la base de datos. En este caso le concederemos todos los permisos y le decimos el host donde se pueden conectar. En este caso como se conectará desde la misma maquina ponemos localhost. (ya se tiene seleccionada la base de datos con el comando anterior)

```
mysql> GRANT ALL ON locationDB.* TO 'adminDB'@'localhost';
```

6)Con esto termina la configuración de la base de datos y podemos cerrarlo mediante el comando exit.

Código:

Para empezar, se ha de volver a hacer la distinción entre la parte de base de datos y la parte de comunicación mediante el protocolo MQTT. Primero se tiene que abrir una conexión con la base de datos a la cual se le han de pasar como parámetros donde se encuentra el servidor, en este caso en localhost, el nombre y la contraseña del usuario

que hemos creado en la configuración de la base de datos, y por último el nombre de la base de datos.

```
db = MySQLdb.connect(mysql_server, mysql_username, mysql_passwd, mysql_db)
```

Las otras partes del código perteneciente a la interacción con la base de datos está dentro del método on_message, que es la definición de la función a la que se llama cuando se recibe un mensaje.

Cuando se recibe un mensaje con los datos, ya sean de temperatura o localizaciones se ha de guardar en la base de datos. Para ello se crea un cursor con el método cursor() con los parámetros anteriormente creados en db. Para acceder a registros o insertar registros en las bases de datos de MySQL se realiza mediante cursores que apuntan a dichas bases y y pueden realizar operaciones con ellas. Una vez tenemos el cursor, sobre él podemos ejecutar el comando execute(). Este comando tiene diferentes modos de emplearse, con un parámetro solo o con dos. Si se realiza con solo un parámetro este ha de ser una sentencia SQL que ejecutar y en caso de tener 2 el primero es también esta sentencia SQL mientras que el segundo son las variables donde está la información a guardar. Con la primera sentencia creamos una tabla, en caso de que no exista, con el nombre locationTABLE y se le especifican los nombres de las columnas de la tabla que corresponden para dar nombre a los diferentes registros dentro de la tabla.

En la segunda ejecución insertamos los datos en esa tabla con los valores de las variables indicadas.

```
with db:
    c = db.cursor()
        c.execute("CREATE TABLE IF NOT EXISTS locationTABLE(Id INT PRIMARY KEY AUTO_INCREMENT, Date VARCHAR(25),Latitude VARCHAR(25), Longitude VARCHAR(25), Time VARCHAR(25))")
        c.execute("INSERT INTO locationTABLE(Date, Latitude, Longitude, Time) VALUES(%s, %s, %s, %s)", (Date, lat, long, time))
```

También hay otra tabla en la que se guardan las temperaturas, y se realiza de la misma manera solo que se le introducen 2 parámetros, dispositivo y temperatura.

Primero se ha de definir la función on_connect que se llamará cuando reciba una respuesta del bróker. Esta función se comprueba si se ha podido conectar con el servidor. La función recibe la instancia para el cliente, la información del usuario, los flags recibidos por el bróker y el resultado de la conexión.

El valor del resultado de conexión (rc) que nosotros recibimos lo comprobaremos para saber si la conexión con el bróker se ha realizado con éxito o no. Si el resultado de la conexión es 0 quiere decir que la conexión se ha realizado exitosamente, en caso contrario no se ha podido conectar con el bróker.

```
def on_connect(client, userdata, flags, rc):
    if rc == 0:    #connected
```

La siguiente función por definir es on_message que es llamada cuando se recibe un mensaje. A esta función se le pasan la instancia del cliente, información del usuario y el mensaje, como parámetros. Del mensaje recibido se puede sacar el tópico con el método topic y el contenido de ese mensaje con el método payload. En nuestro caso usamos el tópico para diferenciar entre los diferentes mensajes y sacamos la información para posteriormente guardarla en la base de datos.

```
def on_message(client, userdata, message):
message.topic
message.payload
```

También se realiza una comprobación en la que si la temperatura recibida es superior a un número máximo la plataforma de gestión pública un mensaje en el tópico alerts para avisar al usuario de que hay un problema con la temperatura. Esto se hace mediante el método publish que requiere un tópico, un mensaje y una calidad de servicio como parámetros.

```
client.publish(topicAlerts,"ESP32A temp. max." + message.payload, qosAlerts)
```

Una vez definidas las funciones se ha de crear la instancia de cliente y establecer en ese cliente el usuario, contraseña con el método username_pw_set y finalmente adjuntar las funciones creadas anteriormente al callback del cliente.

Una vez el cliente está configurado se puede llamar al método connect con la dirección del bróker y el puerto para realizar la conexión.

Mediante el método `client.loop_start()` se crea un thread en segundo plano que llama al método loop que es el encargado de gestionar los eventos de la red en el protocolo MQTT.

Una vez ya se ha conectado al bróker el programa ha de pedir la suscripción mediante el método subscribe que requieres los tópicos y las cualidades de servicio de dichos tópicos. En nuestro caso nos subscribimos a los tópicos location y temperatures, con calidades de servicio 1 y 0 respectivamente

4. Resultados

Uno de los objetivos más importantes de este proyecto era realizar un sistema IoT con el protocolo MQTT en el que se pudiera mostrar el funcionamiento del protocolo. Una vez implementado el sistema IoT, para mostrar su correcto funcionamiento se ha de realizar un análisis de los tipos de mensajes, comunicaciones entre clientes y servidor y funcionalidades del protocolo MQTT.

4.1. Análisis mensajes del sistema

El mensaje connect se especifica la información necesaria para hacer la conexión del cliente con el borker. En el sistema diseñado, la configuración del mosquitto hace necesaria datos de autentificación para conectarse a él, por lo que el cliente los ha de enviar al broker para que este acepte la conexión. En el diseño del sistema se implementa un mensaje will para que si el esp32 se desconecta avise al sistema y este lo notifique. Esta configuración se manda en el mensaje connect del cleinte al broker. Como se puede ver en la imagen la conexión con el broker requiere de un identificador único. Una de las pruebas que se realizo fue conectar 2 clientes con el mismo identificador y el sistema no funciona, ya que el broker necesita el identificador para dirigir los paquetes al los suscritores.

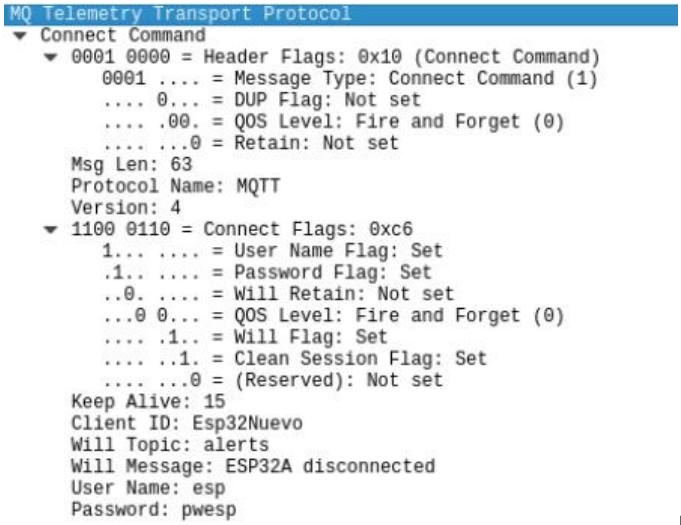


Figura (8)

Los mensajes de conexión requieren respuesta, y en ella se especifica si ha sido aceptada, o no. Y en caso de no serlo te dice el porqué, ya sea contraseña incorrecta, fallo del servidor, etc.

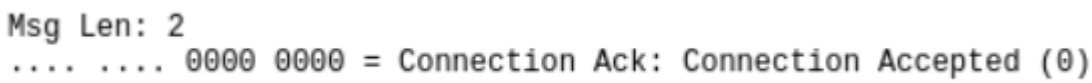


Figura (9)

En referencia al mensaje de publicación hay 2 variantes, que dependen de la calidad de servicio. En caso de tener QoS superior a cero, necesitas un identificador de paquete para que luego los mensajes de confirmación lo incluyan y se pueda seguir el flujo de mensajes. Si la QoS es cero no incluirá identificador de mensaje. Aquí podemos ver un mensaje publicado por el terminal móvil donde se aprecia en la cabecera fija la calidad de servicio 1, y esto implica que tenga identificador de mensaje.

```
MQ Telemetry Transport Protocol
▼ Publish Message
  ▼ 0011 0010 = Header Flags: 0x32 (Publish Message)
      0011 .... = Message Type: Publish Message (3)
      .... 0... = DUP Flag: Not set
      .... .01. = QOS Level: Acknowledged deliver (1)
      .... ...0 = Retain: Not set
    Msg Len: 41
    Topic: location
    Message Identifier: 5
    Message: 41.3888534,2.1119687,19:09:11
```

Figura (10)

En referencia al mensaje de publicación tenemos los mensajes de reconocimiento en caso de que se haya transmitido con QoS 1 o 2. Si se transmite un mensaje con QoS1 se ha de realizar una comprobación simple con el envío de un mensaje de reconocimiento de publicación publish ack. Este mensaje solo contiene la longitud del mensaje y el identificador. Este identificador es mismo que el del mensaje publicador que lo solicita. En el sistema cuentan con respuesta simple (solo ack) los mensajes que publica el móvil al tópico location.

```
Publish Ack
▼ 0100 0000 = Header Flags: 0x40 (Publish Ack)
    0100 .... = Message Type: Publish Ack (4)
    .... 0... = DUP Flag: Not set
    .... .00. = QOS Level: Fire and Forget (0)
    .... ...0 = Retain: Not set
  Msg Len: 2
  Message Identifier: 4
```

Figura (11)

Pero si se ha transmitido con QoS2 la confirmación consta de 3 mensajes, received, reliase y complete. Estos mensajes son iguales que el publish ack. La confirmación de los mensajes de QoS2 presenta el siguiente intercambio de mensajes de confirmación entre cliente y bróker:

Figura (12)

Los mensajes de tipo suscripción se especifican los típicos a los cuales el cliente quiere suscribirse y la calidad de servicio a la que quiere suscribirse.

```
Subscribe Request
▼ 1000 0010 = Header Flags: 0x82 (Subscribe Request)
    1000 .... = Message Type: Subscribe Request (8)
    .... 0... = DUP Flag: Not set
    .... .01. = QOS Level: Acknowledged deliver (1)
    .... ...0 = Retain: Not set
  Msg Len: 26
  Message Identifier: 2
  Topic: temperatures
  .... ..00 = Granted Qos: Fire and Forget (0)
  Topic: alerts
  .... ..10 = Granted Qos: Assured Delivery (2)
```

Figura (13)

El bróker responde a estos mensajes de suscripción con la confirmación de si ha sido aceptado o no para cada uno de ellos, junto con la calidad de servicio de cada uno de ellos. Usa el mismo identificador de paquete que la petición de suscripción.

Por ultimo tenemos el mensaje de desconexión que al igual que los mensajes ping request y ping response está formado solamente por la cabecera fija.

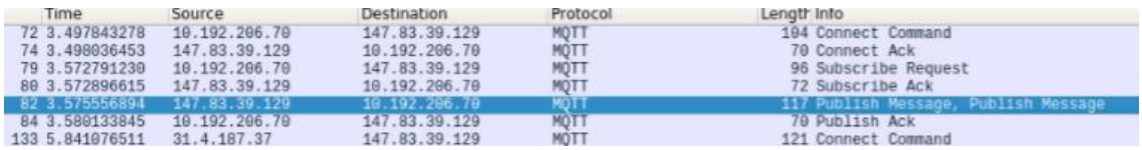
4.2. Análisis funcionalidades

Will message: El ESP32 al realizar la conexión especifica un will message, will tópico, will QoS y will retain, al bróker que enviar en caso que detecte que se ha desconectado. En la imagen del mensaje de conexión podemos ver esa información. El mensaje enviado es tratado como un publish, por tanto, han de ser los clientes los que lo interpreten analizando su contenido y deducir que se ha desconectado. En el sistema diseñado, el móvil, cuando recibe este mensaje en el tópico alertas y ve el mensaje indicando la desconexión muestra por pantalla la advertencia.

Figura (14)

Como vemos en la imagen si compara con un mensaje de publicación normal, con calidad QoS0, no tiene ninguna diferencia, exceptuando del contenido y la longitud.

Retain message: El sistema implementa mensajes retenidos en el bróker. El sistema está diseñado para tener 2 mensajes guardados con esta opción en los tópicos temperatures y location. De esta manera cuando se conecta la plataforma de gestión obtiene dichos mensajes. De temperaturas obtiene el nombre del dispositivo y en el de alerts el día correspondiente. Estos mensajes son enviados justo después de suscribirse, antes de realizar cualquier otra acción.



	Time	Source	Destination	Protocol	Length	Info
72	3.497843278	10.192.206.70	147.83.39.129	MQTT	104	Connect Command
74	3.498036453	147.83.39.129	10.192.206.70	MQTT	70	Connect Ack
79	3.572791230	10.192.206.70	147.83.39.129	MQTT	96	Subscribe Request
80	3.572896615	147.83.39.129	10.192.206.70	MQTT	72	Subscribe Ack
82	3.575556894	147.83.39.129	10.192.206.70	MQTT	117	Publish Message, Publish Message
84	3.580133845	10.192.206.70	147.83.39.129	MQTT	70	Publish Ack
133	5.841076511	31.4.187.37	147.83.39.129	MQTT	121	Connect Command

Figura (15)

Aquí también podemos apreciar un comportamiento de la comunicación a nivel de encapsulado. Cuando el bróker tiene diversa información que enviar algún cliente, puede agrupar los paquetes MQTT en un mismo paquete TCP. De esta manera provoca que la comunicación sea más rápida y fluida.

No solo encapsula 2 paquetes MQTT dentro de uno de TCP en este caso, sino que cuando se somete el sistema a mucho tráfico de paquete, encapsula los que sean necesarios.

En el sistema los mensajes al tópico alerts se manda con dos tipos de cualidades de servicio diferentes. El ESP32 lo manda con QoS0, la plataforma de gestión lo manda con QoS2. El terminal móvil comunica al bróker que se suscribe con QoS2. El bróker recibe el mensaje del ESP32 con QoS 0, y al enviarlo lo replica con la misma QoS 0 aunque el dispositivo móvil se haya suscrito a QoS2. Por lo que se deduce que al hacer la subscripción la cualidad de servicio indicada es la máxima posible, no la requerida. Porque el bróker, podría recibir en QoS0 del ESP32, y enviarlo al móvil con QoS2 y realizar solo las comprobaciones con él.

Clean seasion Es un flag que está en el mensaje de conexión. Nuestro terminal móvil tiene este flag a 0, por lo que mantienen una conexión persistente con el bróker. Esto provoca que, al desconectarse, si alguien alguien publica al tópico del cual está suscrito el bróker lo conserva hasta que se vuelve a conectar. Pero solo en el caso que los mensajes sean con calidad de servicio superior a 0. En la imagen de abajo podemos ver como al volver a conectarse porque había perdido la conexión, el bróker le manda el paquete que le per tocaba, con el dup flag activo.

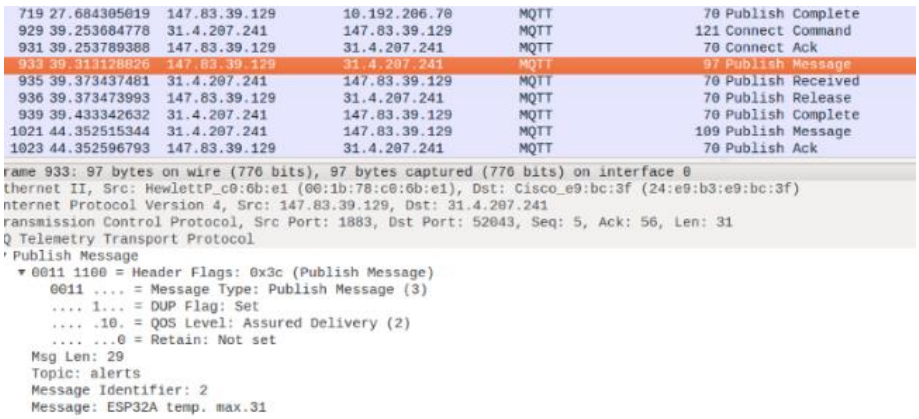


Figura (16)

4.3. Base de datos

En la plataforma de gestión está la base de datos con las 2 tablas donde se guardan los datos de temperatura y de localizaciñon. Aquí tenemos un ejemplo.

```
mysql> SELECT * FROM locationTABLE;
+----+------------+-----------+-----------+----------+
| Id | Date       | Latitude  | Longitude | Time     |
+----+------------+-----------+-----------+----------+
|  1 | 22/01/2018 | 41.388856 | 2.111971  | 21:09:17 |
|  2 | 22/01/2018 | 41.3888293 | 2.1119882 | 21:09:28 |
|  3 | 22/01/2018 | 41.3888561 | 2.1119695 | 21:09:38 |
|  4 | 22/01/2018 | 41.3888561 | 2.1119695 | 21:09:48 |
|  5 | 22/01/2018 | 41.3888542 | 2.1119669 | 21:09:59 |
|  6 | 22/01/2018 | 41.388857  | 2.1119723 | 21:10:49 |
|  7 | 22/01/2018 | 41.3888552 | 2.1119712 | 21:10:59 |
|  8 | 22/01/2018 | 41.3888439 | 2.1119702 | 21:11:09 |
|  9 | 22/01/2018 | 41.3888439 | 2.1119702 | 21:11:19 |
| 10 | 22/01/2018 | 41.3888564 | 2.1119695 | 21:11:30 |
| 11 | 22/01/2018 | 41.3888571 | 2.1119708 | 21:11:41 |
| 12 | 22/01/2018 | 41.3888488 | 2.1119808 | 21:11:51 |
| 13 | 22/01/2018 | 41.388856  | 2.1119714 | 21:12:02 |
| 14 | 22/01/2018 | 41.3888534 | 2.1119677 | 21:12:12 |
| 15 | 22/01/2018 | 41.3888534 | 2.1119677 | 21:12:22 |
| 16 | 22/01/2018 | 41.3888555 | 2.1119695 | 21:12:32 |
| 17 | 22/01/2018 | 41.3888741 | 2.111969  | 21:12:42 |
| 18 | 22/01/2018 | 41.388857  | 2.1119722 | 21:12:52 |
+----+------------+-----------+-----------+----------+
18 rows in set (0,00 sec)
```

Figura (17)

```
    -> ;
+----+--------+-------------+
| Id | Device | Temperature |
+----+--------+-------------+
|  1 | ESP32A | 26          |
|  2 | ESP32A | 26          |
|  3 | ESP32A | 26          |
|  4 | ESP32A | 26          |
|  5 | ESP32A | 26          |
|  6 | ESP32A | 26          |
|  7 | ESP32A | 26          |
|  8 | ESP32A | 26          |
|  9 | ESP32A | 26          |
| 10 | ESP32A | 26          |
| 11 | ESP32A | 26          |
| 12 | ESP32A | 26          |
| 13 | ESP32A | 26          |
| 14 | ESP32A | 26          |
| 15 | ESP32A | 26          |
| 16 | ESP32A | 26          |
| 17 | ESP32A | 26          |
| 18 | ESP32A | 26          |
| 19 | ESP32A | 26          |
| 20 | ESP32A | 26          |
| 21 | ESP32A | 26          |
| 22 | ESP32A | 26          |
| 23 | ESP32A | 26          |
| 24 | ESP32A | 26          |
```

Figura (18)

4.4. Pruebas

Se han realizado pruebas de estrés sobre el sistema. En este no actuaba el cliente ESP32 puesto que tiene la limitación del sensor que solo puede captar información como mínimo a 1 segundo. Y se ha añadido un cliente publicador situado en el mismo ordenador que la plataforma de gestión. Este publicador enviaba 6000 mensajes cada 10 ms, en QoS0, QoS1 y QoS2. En estas pruebas se puede comprobar que muchos de los mensajes destinados al terminal móvil encapsulan varios mensajes MQTT dentro de un mismo paquete TCP. En el teléfono móvil se contaban los mensajes recibidos. Con las diferentes QoS siempre llegaban todos los paquetes. Se realizaron las pruebas tanto en 4G como en 2G, y el resultado fue el mismo. El protocolo TCP se encarga de que si hay algún mensaje erróneo retransmitirlo.

De estos test se concluye que las calidades de servicio se han de implementar en casos en los que las condiciones de la red sean bastante malas y con tasa de perdida de paquetes muy alta. Porque de no ser así solo se consigue sobrecargar el bróker.

5. Costes

Antes de realizar los cálculos cabe destacar que estos son orientativos. No todos los ingenieros juniors cobran igual, los elementos usados para bróker y plataforma de gestión ya estaban a nuestra disposición por lo que se ha hecho un cálculo de cuanto podrían costar.

Horas/día	Dias/semana	semanas/mes	meses	total horas
5	5	4	4	400

Sueldo junior neto por hora	8	€/h
Precio final mano de obra	3200	€

En este proyecto se ha planteado 2 posibilidades con tal de abaratar una de ellas.

Usando un ordenador de sobremesa para el bróker:

Precio final mano de obra	3200	€
Bróker (ordenador sobremesa)	450	€
ESP32	18,99	€
DHT11	3,14	€
Plataforma gestión (ordenador portátil)	550	€
Router	50	€
Smartphone	140	€
total	4412,13	€

Usando una Raspberry pi 2 como bróker:

Precio final mano de obra	3200	€
Bróker (Raspberry)	34	€
ESP32	18,99	€
DHT11	3,14	€
Plataforma gestión (ordenador portátil)	550	€
Router	50	€
Smartphone	140	
total	3996,13	€

*Nota: no se incluye el

El factor que aumenta el valor del proyecto es el del trabajo realizado por el ingeniero. Teniendo en cuenta que este documento es un demostrador para que terceras personas o empresas lo puedan tomar como ejemplo, el coste de la realización eliminaría los costes derivados de la mano de obra. Pero si por el contrario se encarga a alguna empresa que te lo realice sumaría esta cuantía total de 4412,13€ o 3996,13€.

6. Conclusions and future development:

Una vez finalizado el proyecto podemos concluir que se han alcanzado los objetivos marcados al inicio de este. Se ha desarrollado un sistema IoT, cumpliendo los requisitos previamente establecidos. En el sistema se muestra las funcionalidades, características y flujos de comunicación de mensajes entre dispositivos del protocolo MQTT. Por esta razón este documento puede servir de base para realizar un diseño, total o parcial, de un sistema IoT, tanto a nivel de formación, profesional o particular. Parte de este proyecto ha sido utilizado por mi supervisora para la impartición de unas clases prácticas de una asignatura del master, demostrando que se puede ser útil a nivel universitario.

También se comprueba la gran aplicabilidad del protocolo MQTT a distintos entornos, tecnologías y lenguajes de programación. Se ha creado un cliente en un microcontrolador ESP32, que es un dispositivo pensado para el mundo del IoT. Otro cliente se ha implementado en un Smartphone con Android, siendo este un dispositivo que mucha gente tiene acceso a él. Y el ultimo cliente se ha implementado usando Python. No solo es a nivel de cliente, sino que el bróker ha podido ser programada en el entorno Ubuntu y en una Raspberry con Raspbian.

En ámbito personal, la realización de este proyecto ha sido muy enriquecedora, ya que me ha permitido conocer otros lenguajes de programación que no había estudiado durante mis estudios como son Andorid y arduino. También he aprendido a trabajar con bases de datos, ya que en ningún proyecto de mi carrera utilicé. Y para finalizar me ha descubierto un gran interés por el mundo del IoT y por el que seguiré trabajando a nivel personal o profesional.

Bibliography:

1] Andrew Banks, Rahul Gupta,-MQTT Version 3.1.1 OASIS Standard – 29/10/2014
2] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, Jesus Alonso-Zarate, “A Survey on Application Layer Protocols for the Internet of Things” Tecnologic de Telecomunicacions de Catalunya, 2015
3] Antti Luoto, Kari Systa, “IoT Application Deployment Using Request-response Pattern with MQTT”, Tampere University of Technology, Tampere, Finland
4] Makkad Asim, “A Survey on Application Layer Protocols for Internet of Things (IoT)” , 2017
5] Daniel Happ, Adam Wolisz “Limitations of the Pub/Sub Pattern for Cloud Based IoT and Their Implications”, Technische Universit¨at Berlin, Telecommunication Networks Group (TKN)

ANEXO Instalaciones

MOSQUITTO

A.1 Instalación mosquitto versión 1.4.14

La instalación se realiza mediante comandos desde el terminal:

1)Primeramente se ha de agregar los repositorios necesarios para la instación de mosquitto.

```
sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa
```

2)Antes de realizar cualquier instalación es conveniente comprobar y actualizar el el sistema operativo Ubuntu para que la compatibilidad sea mayor.

```
sudo apt-get update
```

3)Instalar mosquitto broker

```
sudo apt-get install mosquitto
```

4) Instalar cliente MQTT para poder realizar tests

```
sudo apt-get install mosquitto-clients
```

5) Después de realizar esto se iniciará directamente el servicio, y deberemos pararlo e iniciarlo otra vez para que pueda funcionar ya que mosquitto es carga su configuración cada vez que se inicia el servicio.

```
sudo service mosquitto restart
```

Se puede usar también la función con el comando stop y luego start.

ARDUNIO

Instalación arduino IDE para Windows:

1) Primero tenemos que descargar desde la página oficial el ejecutable a instalar. Y procedemos a su instalación

https://www.arduino.cc/en/Main/Software (Para la realización del proyecto se ha usado la versión 1.8.4, pero no debería haber problemas si son superiores)

2) Una vez instalado, necesitamos descargar los contenidos que arduino pueda interactuar con el hardware del ESP32.

   https://github.com/espressif/arduino-esp32/archive/master.zip

3) Al terminar la descarga debemos descomprimir el archivo y copiar los elementos que contiene en la carpeta esp32. Esta carpeta se encuentra en dentro dela carpeta arduino, hardware y si no existen deberíamos crear las carpetas espressif y esp32 tal y como indica en la ruta anterior.

   C:\Users\"my usuario"\Documents\Arduino\hardware\espressif\esp32

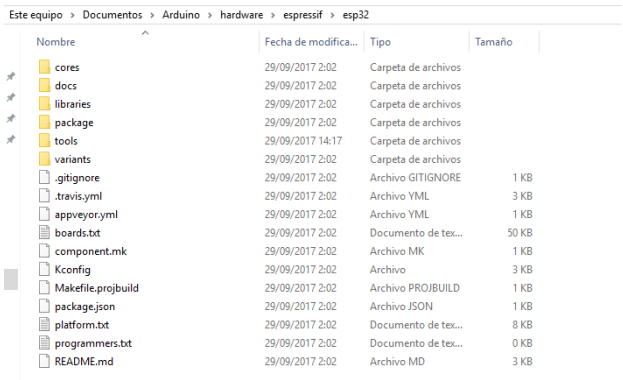


Figura (19)

4) Una vez añadido el repositorio es necesario para que arduino pueda interactuar con nuestro esp32, se ha de instalar el compilador Xtensa GNU compiler collection (GCC). Esto se hará fácilmente ejecutando el archivo get.exe que hay en la carpeta tools que hemos añadido antes en esp32.

Con esto habríamos terminado la instalación arduino IDE y añadido los elementos necesarios para poder actuar con el esp32

Conexión esp32 y configuración arduino:

1) Una vez instalado, conectamos el esp32 y Windows se encarga de instalar los drivers necesarios para poder comunicarse con la placa.

2) Posteriormente se inicia arduino para realizar la configuración requerida.

3) El siguiente paso es indicar al programa que puerto serie vamos a utilizar. Esto se encuentra en herramientas, puerto y se selecciona el puerto COM en el dispositivo está conectado.

4) Por último, queda indicar al programa que tipo de elemento está conectado a ella. Por tanto, en herramientas, placa, y seleccionamos el ESP32 dev module.